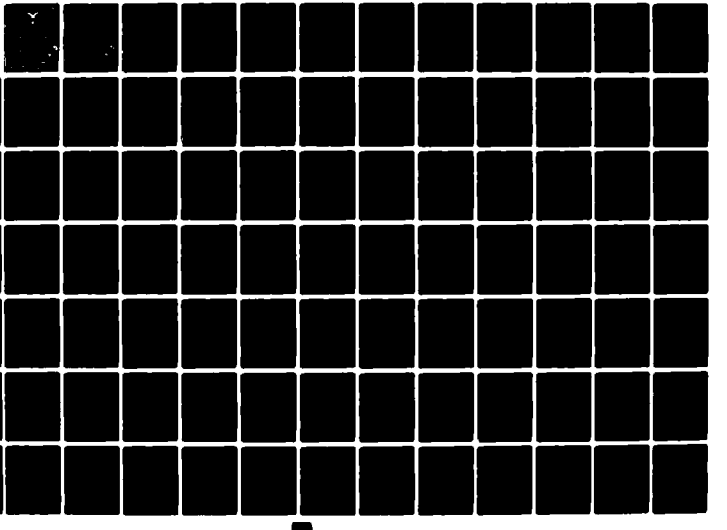
AD-A115 553 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM.(U)
DEC 81 N J MURPHY
UNCLASSIFIED AFIT/GCS/MA/81D-5 NL



UNITED STATES AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/MA/81D-5

MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM

THESIS

AFIT/GCS/MA/81D-5 Nancy J. Murphy
Captain USAF

Approved for Public Release; Distribution Unlimited

AFIT/GCS/MA/81D-5

MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements of the Degree of
Master of Science

by

Nancy J. Murphy, B.S.
Captain USAF
Graduate Computer Science

December 1981

Preface

This paper documents the procedures, considerations, and design decisions made, leading to the implementation of MODIFY, a machine-independent computer program designed to assist programmers in maintaining source files. MODIFY is written in standard FORTRAN 77.

My sincere thanks are extended to four people who supported me during the past nine months. First, my sincere thanks to Mr. Charles W. Richard, Jr., my faculty advisor, for his timely guidance, persistence, and constructive criticism. Second, my thanks to Mr. William K. McQuay, my sponsor, for his support and guidance throughout the duration. Third, my thanks to Mrs. Linda L. Schafer, my typist, for her dedication and attention-to-detail in typing this report. And last, but definitely not least, my most sincere appreciation and indebtedness goes to my husband, Roger, for his patience, understanding, and support throughout... I love you Roger, and now you are #1 again.

Nancy J. Murphy

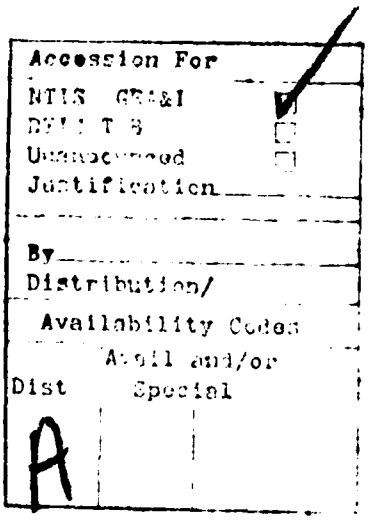


Table of Contents

List of Figures

List of Tables

Abstract

It is not uncommon for an organization to maintain a set of files on two or more different computer systems. Each system has its own way of allowing the user to make changes to the files, forcing users not only to know those unique procedures but to make duplicate changes for each system. Making the same changes to the same files on different computer systems requires additional time and increases the risk of making errors. This paper presents the procedures, considerations, and design decisions made, leading to the implementation of MODIFY, a machine-independent batch utility designed to assist programmers in maintaining source files. MODIFY, which is written in standard FORTRAN 77, handles routine update/retrieval functions and provides a complete audit trail of changes.

MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM

I. Introduction

The purpose of this project is to design and implement a machine-independent software package similar to the Control Data Corporation (CDC) UPDATE program, a batch utility capable of creating, updating, and retrieving compressed source files in a program library. This chapter will provide the reader an explanation of why such a package is needed, the subset of CDC's UPDATE which has been emulated, and the approach taken in accomplishing the design and implementation of the machine-independent package along with an overview of the remaining chapters of this paper.

Background

The emulation of CDC's UPDATE utility is needed by personnel from the Analysis & Evaluation Branch, Electronic Warfare Division, Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio (AFWAL/AAWA).

The AAWA personnel are required to maintain and execute application programs on two different computer systems. The primary computer used is a DEC VAX 11/780, but when this computer is processing classified information, a CDC Cyber 74/750 computer system must be used. Each system has its own way of allowing the user to make changes to source files, forcing users not only to know those unique

procedures but to make duplicate changes for each system. Making the same changes to the same files on two different computers requires additional time and increases the risk of making errors.

Problem and Solution

A machine-independent program is needed allowing the user to make only one set of changes for source files stored on both the DEC VAX 11/780 and the CDC Cyber 74/750 computer systems. The Avionics Laboratory personnel are familiar with and have used a subset of CDC's UPDATE utility in the past. Therefore, a machine-independent program emulating a subset of CDC's UPDATE utility is provided.

Scope

The machine-independent update utility developed, called MODIFY, was designed in such a way to recognize eleven UPDATE directives (i.e., eleven commands) and three additional directives (commands) not available in UPDATE. The eleven UPDATE directives supported are the most commonly used and are:

*ADDFILE
*CALL
*COMDECK
*COMPILE
*DECK
*DELETE
*IDENT
*INSERT
*RESTORE
*SEQUENCE
*YANK

Once MODIFY was implemented on the target machine, DEC VAX 11/780, the AAWA personnel could use the same update deck (using the UPDATE subset) on either computer system. When (or if) MODIFY is implemented on the CDC Cyber 74/750 system, the full complement of MODIFY directives will be available on both systems, including the three directives not on UPDATE:

*EDIT
*FILL
*SCAN

Approach and Presentation

An extensive study was conducted of CDC's UPDATE Reference Manual to emulate a subset of its capabilities on the target machine, the DEC VAX 11/780. Seven other utility packages were reviewed in an effort to include the most attractive capabilities (even more than what UPDATE offers) and to gain insight into the file structures needed to accomplish the tasks. Once all the directives and file structure decisions were made, MODIFY was designed, coded, tested, and implemented on the DEC VAX 11/780. MODIFY is coded in American National Standards Institude (ANSI) standard FORTRAN 77, promoting machine independence.

The remaining chapters of this report describe the steps taken to design and implement MODIFY. Chapter II outlines the characteristics and capabilities of already existing utilities and then describes those capabilities selected to be included in MODIFY. Chapter III describes the decisions and assumptions made during the development of

the program and an overview of the program structure. Chapter IV ties it all together describing the results of this project and recommendations to enhance MODIFY.

II. Detailed Analysis

A certain amount of research and several decisions needed to be made before the actual design of the program was started. Since this project evolved around an already existing program, the process may seem to be trivial at first thought. However, FORTRAN has traditionally been a "computational" language and placed certain limits on this "data-manipulating" project. FORTRAN was selected, however, because of its popularity, i.e. most computer systems have a FORTRAN compiler. Further, of the eight utilities studied, the manuals available were programmer-guide-type manuals; giving detailed descriptions on what the utility does and what the user needs to do to use the package, but very little on how the utility actually handles the tasks.

Since MODIFY, in part, is a subset of CDC's UPDATE utility, the first section of this chapter gives an overview of UPDATE's features, type of runs, directives, file structures, and control card parameters. The most useful directives of UPDATE have already been identified and are included in MODIFY; however, other utility packages have popular features not available in UPDATE. The second section of this chapter describes the unique capabilities and functions of seven other utility packages. The last section summarizes the characteristics of MODIFY, based on the results of analyzing the eight utility reference manuals.

UPDATE Characteristics

UPDATE is a utility package capable of updating and/or retrieving sets of data stored in one file, called a program library (PL). A program library contains "decks", "comdecks", and "idents". A "deck" usually consists of a source program, but may contain job control language, subprograms, data, etc. A "comdeck", or common deck, contains source code which may be inserted into decks or other comdecks. An "ident" is a correction set which identifies directives and source-code records used in updating a deck or comdeck. The rest of this section outlines UPDATE's capabilities and file structures, the details of which may be found in Appendix A.

Features (Ref 4:1-2). UPDATE has fourteen features which are explicitly listed in Table A-I. The features are not discussed in detail since the descriptions are self-explanatory. This information was used as an initial step in deciding MODIFY's capabilities.

Types of Runs (Ref 4:1-3,1-4). When UPDATE is executed, one of the three modes of operation is either implied by the order of the directives or explicitly selected on the UPDATE job control card. The type of runs are: creation, correction, or copy.

The basic rule for distinguishing a creation run from a correction run is: a creation run is implied when UPDATE does not encounter any directives prior to encountering a

*DECK or *COMDECK directive. It is a "basic" rule because there are ten directives which may precede a *DECK or *COMDECK. However, the ten directives are the "hardly-ever-used" directives and would add little to this discussion.

A correction run usually begins with an *IDENT directive but several other directives are permissable. A one-time-only creation run must establish the program library before any correction or copy runs may be executed.

A sequential-to-random copy run may be requested by including an "A" as a parameter on the UPDATE job control card. A random-to-sequential copy run is requested by including a "B" on the UPDATE job control card.

Directives (Ref 4:2-1 thru 2-30). UPDATE recognizes forty-one directives, but eleven of them can handle any routine update or retrieval function a user may desire. Many of the directives fall under the "hardly-ever-used" category while others allow a "different-way-of-doing-it" option. For example, to insert a new data-card record the *BEFORE directive or the *INSERT directive can be used. Table A-II lists the UPDATE directives along with an acceptable abbreviation and a brief description of each one.

File Structures (Ref 4:3-1 thru 3-15, C-1 thru D-2). UPDATE optionally interacts with six user-accessible files and six scratch files. Figure 1 depicts this relationship and identifies the standard name of these files.

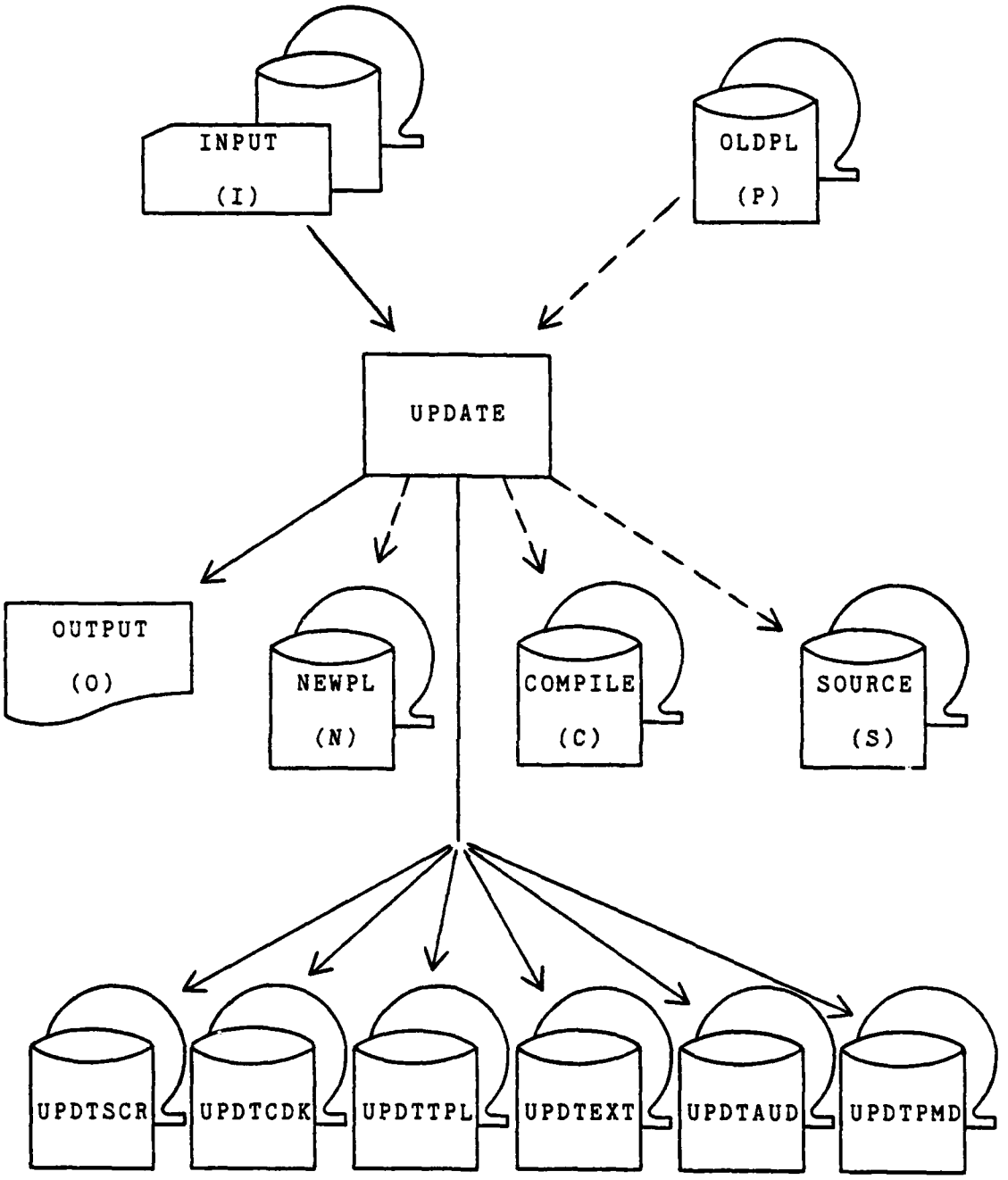


Figure 1. UPDATE File Interaction

The dotted lines indicate optional files, depending on the type of run. The alphabetic character in parenthesis indicates the UPDATE control card parameter (explained in the next section) which can be used to change the default name of the user-accessible files. With the exception of a creation run, OLDPL is the latest version of NEWPL generated from a previous UPDATE run. The characteristics of these twelve files are listed in Table A-III.

The program library contains compressed card images. UPDATE compresses a card image by replacing three embedded spaces with 0002, four embedded spaces with 0003,..., 67 embedded spaces with 00770002_8, etc. Trailing spaces are not considered as embedded and are not included in the card image. This reduces the length of most records, thereby decreasing the amount of space required to store the program library.

The formats of the records generated by UPDATE for both random and sequential program libraries are available in the UPDATE Reference Manual. The description of these records is not repeated here since the formats were not utilized in MODIFY. Much of the information involves bit accessibility, a level FORTRAN 77 cannot handle, and control information needed to support program libraries created by older versions of UPDATE.

Control Card Parameters (Ref 4:4-2 thru 4-10). UPDATE is invoked by including an "UPDATE,p1,p2,...,pn." control card in the job control stream, where p1, p2,..., pn are

optional parameters used to specify modes and files for the run. Table A-IV lists the parameters and their functions. The List (L) parameter allows the user to specify output listing options to be invoked. Table A-V lists the acceptable values and a brief description of what each value will generate. The default value for a creation run is A12, for a correction run is A1234.

Literature Review

Several UPDATE-type utility packages are available. Seven such packages were studied in detail in order to acquire additional information. This section briefly outlines the capabilities of the seven utilities, as they differ from UPDATE.

EDIT (Ref 3) apparently is an early version of UPDATE, but not nearly as diversified. The thirteen directives available are a subset of UPDATE, all the files are sequential, and all the unit information referred to tape drives.

HISTORIAN (Ref 8) is a machine-independent, commercially-available package supporting twenty-one directives in which all but one make up a subset of UPDATE. The one directive not supported by UPDATE is *SCAN, or *SC, and allows for string search and replacement. The functions performed by the UPDATE control card parameters are handled via a HISTORIAN card, the first card before any directives in the input file. Decks or comdecks containing more than 4095 records must be separated so not to exceed the

4095-record limit. HISTORIAN also has a "salvage" run which attempts to recover program libraries after a machine malfunction.

LIBRARIAN (Ref 1) is an IBM-oriented package offering three attractive directives *not* available in UPDATE. The "-EDIT" performs the string search and replacement function, while "-SCAN" performs a string search and print function, with no replacement. The "-FILL" allows a string of characters to be inserted into specific columns, overlaying the previous contents. A password may be included with each deck to minimize the possibility of inadvertently selecting and updating the wrong deck. A program library may contain (only) up to twenty modules.

MAWLOGS (Ref 7) is a fairly primitive package with only four directives: add text, delete text, add module, and delete module. The directives must be in alphabetic order by deck name. All files are sequential, and they contain 80-character card images. Any time a deck is altered it is resequenced automatically, causing line numbers to change from one update to the next. The package includes a complete listing of the FORTRAN IV program.

MEDIT (Ref 6) is a limited machine-independent package and is neither execution-time nor storage efficient. A "base module" is identified as a sequential file of card images. A "test module" is the result of a run, but an updated version is not saved. As additional changes of the base module are required, the directives are added to the

update card deck and all previous changes are accomplished over (and over) for each succeeding run, until all necessary changes are realized. Once the module is ready for production, an updated version is saved but the history of changes is lost, because the module is automatically resequenced. The package only supports five functions: an insert, a deletion, placement of a "C" in column 1, placement of a space in column 1, and replacement of a string. The package is primitive, but the manual does include a complete listing of the FORTRAN IV program.

PANVALET (Ref 9) is an IBM-oriented package with many functions not included in UPDATE. However, all but one of these unique functions pertain solely to IBM nomenclature. The one unique directive which is not machine dependent is the "EJECT" command which causes the output print file to advance to the top of the next form.

SAIL (Ref 5) is a machine-independent package which has similar capabilities as UPDATE but accomplishes its tasks in an altogether different manner. To outline SAIL's unique functions would be beyond the scope of this report.

MODIFY Characteristics

This section describes the capabilities and file structures of the new utility, MODIFY, the result of studying UPDATE and the seven other packages previously mentioned.

MODIFY is a machine-independent update utility modeled after CDC's UPDATE utility. Like UPDATE, a program library

is maintained which contains decks, comdecks, and idents. The user supplies a MODIFY control card and a set of MODIFY directives and/or data-card records to a MODIFY run. The supplied information is used by MODIFY to create, update, and/or retrieve the specified decks.

Features. The features of MODIFY are listed in Table I along with a brief explanation of how the user requests the features. The differences of MODIFY features as compared to UPDATE features are described in the next paragraph.

MODIFY does not support sequential program libraries; therefore, MODIFY cannot copy libraries from sequential to random format and vice versa, as does UPDATE. Sequential libraries would not be execution nor storage efficient and UPDATE probably has the feature to support environments where magnetic tapes are prominent, rather than disk packs, and to support previous versions of UPDATE. MODIFY does not have the merging of two program libraries because of the way MODIFY processes correction sets, i.e., ident entries cannot be moved once they are entered into the program library directory. MODIFY recognizes abbreviated forms of directives but does not include the capability of turning off the search for the abbreviated forms. This feature was not included since MODIFY contains so few directives; however, if the user wishes to speed up processing (the reason UPDATE has this capability) the user simply needs to use the full directive name. The checksumming of a

Table I

MODIFY Features

Feature	Explanation
- Creation of a PL from source decks	No OLDPL identified and first directive is either *DECK or *COMDECK
- Updating of source decks by inserting, deleting, and restoring cards according to sequence in the deck or according to correction set	Insert via *INSERT, delete via *DELETE, restore via *RESTORE
- Ability to completely and permanently remove correction sets from the program library	A *DELETE followed by a *SEQUENCE, or generate a SOURCE file to be input to a MODIFY creation run
- Generation of a compile file containing corrected output acceptable as input to other processing programs	Any deck updated or specified on *COMPILE will be written to the COMPILE file
- Processing of directives, new text, and new source decks from a file other than the job input file	An "I=filename" specified on the MODIFY card or an alternate file name specified on the *ADDFILE
- Production of fresh source decks from the PL	Include "S" on the MODIFY card
- Generation of a new, updated PL	Include "N" on the MODIFY card
- Comprehensive list output noting any changes occurring during the run and status of PL	See Table VII for all the list options
- Ability to change the directive card master control character	Include "*=value" on the MODIFY card, where "value" is the new control character

Table I, Con't

MODIFY Features

Feature	Explanation
- Recognition of abbreviated forms of directives	MODIFY will automatically accept the abbreviation form or the long form
- Ability to use full 64-character set, including the colon	Machine-dependent but ANSI standard is A to Z, 0 to 9, and the thirteen special characters =+-*/(),.$': and blank

program library deals with sequential program libraries and is not included in MODIFY.

Types of Runs. MODIFY supports only the creation and *correction runs, not the copy run.* A creation run is implied if the first directive is either a *DECK or a *COMDECK. A correction run is implied if the first directive is either an *IDENT, *ADDFILE, *COMPILE, or *SCAN directive. An *IDENT directive must be encountered before any of the eight update-type directives are encountered. The UPDATE copy run is not supported because MODIFY is not capable of maintaining a sequential program library.

Directives. MODIFY recognizes fourteen directives capable of handling routine update or retrieval functions. Table II lists the MODIFY directives along with an acceptable abbreviation and a brief description of each one. The asterisk preceding each name is the default directive card master control character and identifies an input file record

Table II

MODIFY Directives

Name	Abbreviation	Function
*ADDFILE	*AF	Add a deck or comdeck to the PL
*CALL	*CA	Replaces this directive with the contents of the comdeck identified when the deck/comdeck is written to COMPILE
*COMDECK	*CD	Introduces a common deck, common code to be inserted into a deck or another comdeck when CALLed
*COMPILE	*C	Specifies decks to be written to file COMPILE for later compilation
*DECK	*DK	Introduces a source deck, usually contains a source program
*DELETE	*D	Specified cards are flagged as inactive and optionally replaced with cards following the DELETE
*EDIT	*ED	String search and replacement
*FILL	*FI	Insert string of characters into specified columns
*IDENT	*ID	Introduces a correction set
*INSERT	*I	Adds the cards following the INSERT after the specified location
*RESTORE	*R	Reactivates specified cards and optionally inserts the cards following the RESTORE
*SCAN	*SC	String search and print
*SEQUENCE	*S	Resequences active cards and purges inactive cards in specified deck/comdeck
*YANK	*Y	Deactivates all cards of specified correction set, restores decks/comdecks to previous state

as a directive rather than a data card. The *EDIT, *FILL, and *SCAN directives are not supported by UPDATE, but are directives taken from LIBRARIAN (Ref 1).

File Structures. MODIFY optionally interacts with six user-accessible files and five scratch files. Figure 2 depicts this relationship, the dotted lines indicating optional files, depending on the type of run. The diagram does not show default file names for the user-accessible files because these names have not been permanently assigned in the MODIFY program. The names are to be supplied prior to implementing MODIFY, allowing the file names to take on the format used for any given system. For example, the default file name for the card reader on the DEC VAX 11/780 is SYS$INPUT, on the CDC Cyber 74/750 is INPUT, and on an IBM 360 is SYSIN. However, to avoid any confusion which may occur when discussing the files and their names, the UPDATE default file names (INPUT, OUTPUT, OLDPL, NEWPL, COMPILE, and SOURCE) will be used throughout this report. The alphabetic character in parenthesis indicates the MODIFY control card parameter (defined in the next section) used to change the default file name of the user-accessible files.

With the exception of a creation run, OLDPL is the latest version of NEWPL generated from a previous MODIFY run. The characteristics of the MODIFY files are listed in Table III. All but two of the files are defined as formatted, sequential files and no more explanation seems

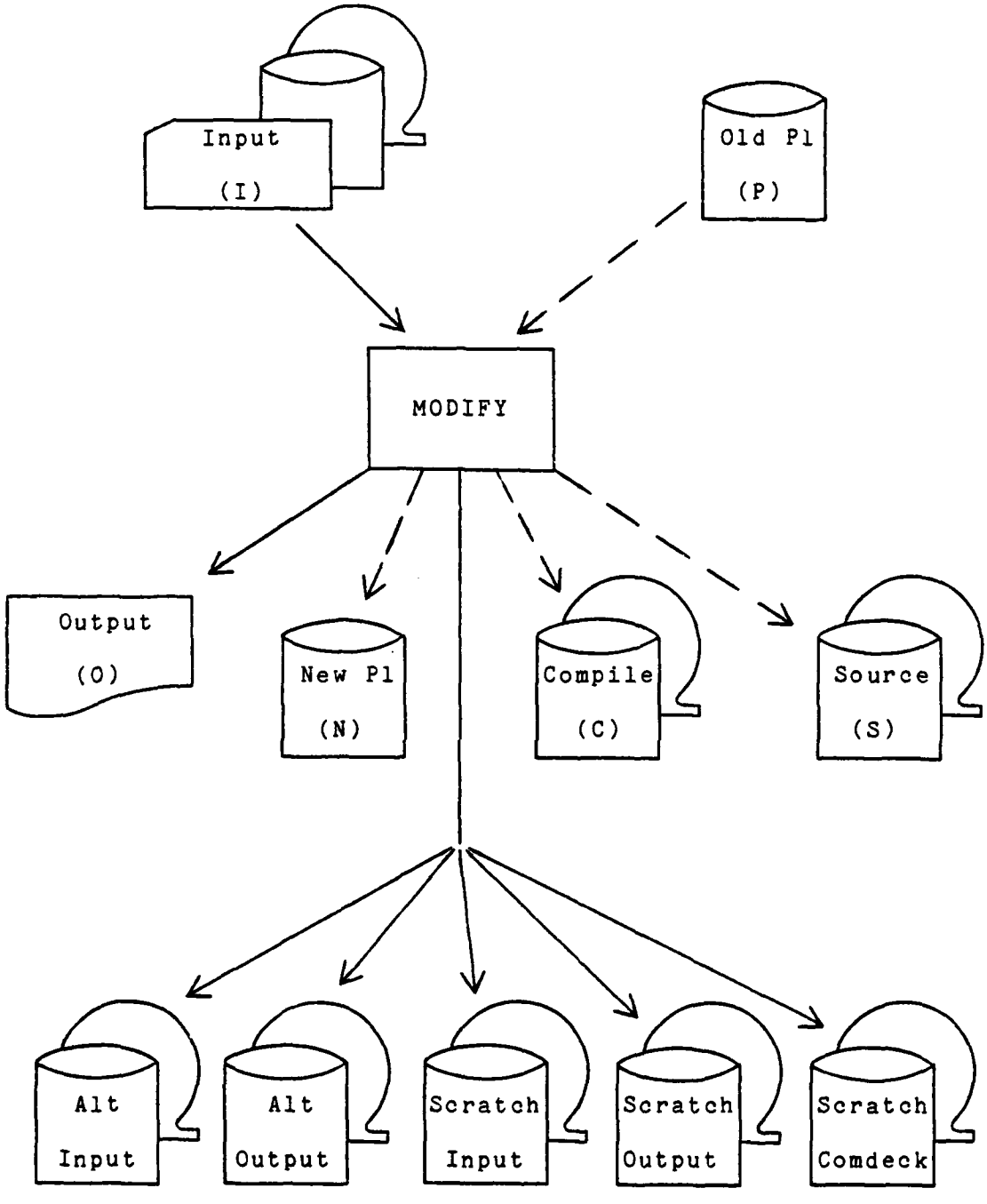


Figure 2. MODIFY File Interaction

Table III

MODIFY File Characteristics

File	Type	Storage	Record Length	Function
Old PL	Unformatted	Direct, Compressed	User-Supplied	Contains old PL
New PL	Unformatted	Direct, Compressed	User-Supplied	Contains new PL
Input	Formatted	Sequential	80	Contains MODIFY card and can contain directives and data cards
Output	Formatted	Sequential	132	Contains listings requested via the "L" control card parameter
Compile	Formatted	Sequential	80	Contains executable program statements suitable for compilation
Source	Formatted	Sequential	80	Contains "active" card images
Alternate Input	Formatted	Sequential	80	Used as the alternate input file if specified
Alternate Output	Formatted	Sequential	132	Used as the alternate output file if specified
Scratch Input	Formatted	Sequential	80	Contains update-type directives encountered on the input file and data cards following
Scratch Output	Formatted	Sequential	132	Contains results of requesting list options 7 and 8

Table III, Con't

MODIFY File Characteristics

File	Type	Storage	Record Length	Function
Scratch Comdeck	Formatted	Sequential	80	Used to hold common decks for later expansion of *CALL cards

necessary. However, several decisions needed to be made before deciding the exact composition of the program library. An extensive discussion of the characteristics considered follows.

The program library file could be sequential- or direct-access (ANSI standard), formatted or unformatted, and contain compressed (suppress preceding spaces, drop succeeding spaces) or uncompressed data cards (store 80-byte cards). In order to minimize execution time and storage requirements, the decision to have a direct-access file organization with compressed cards was made early-on.

The contents of the program library file consists of fixed-length records, as required for direct-access files, in which one of the records is used as the directory (REC #1) and at least one record is generated for each deck/comdeck introduced (REC #2, REC #3, etc). If the input-data cards making up one deck/comdeck are more than will fit in one program library record, MODIFY will automatically start a continuation record (or records) until all

the input-data cards have been processed. In other words, the contents of a deck/comdeck may span over multiple records as depicted in Figure 3. As the diagram shows, the directory is always REC #1 and any decks introduced may be stored in multiple records.

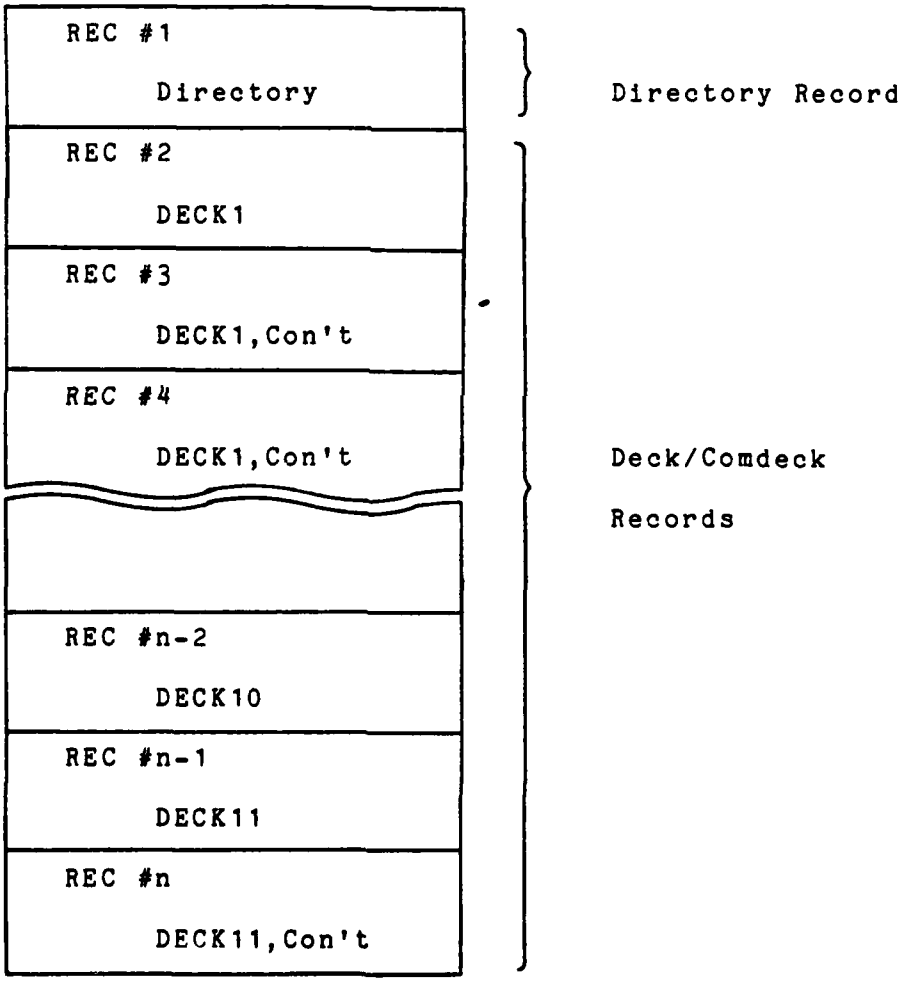


Figure 3. MODIFY Program Library Structure

The directory record is made up of a fixed-length portion followed by a variable-length portion, the contents of which is outlined in Table IV. The first two elements are included only once, whereas at least one set of the last four elements is needed for each deck/comdeck/ident introduced to the PL. The set of four elements makes up what is called a "directory entry". As previously explained, multiple entries may be generated when the contents of a deck/comdeck will not fit in one deck/comdeck-type record. MODIFY will automatically generate a directory entry for each continuation record. An ident (identifying a correction set) may also cause multiple directory entries to be generated. The input-data cards introduced by an ident are not placed in a separate record but are inserted into the deck/comdeck record(s) specified. The ident directory entry is used to keep track of the decks/comdecks updated; that is, the "REC number" element contains the number of the first record of the deck/comdeck updated. The linked-list structure (the fourth element of each directory entry) is used to chain the entries together to keep track of all records updated by the ident. A more detailed description of the directory record format is presented later in this section.

Each deck/comdeck record in the file contains a variable number of variable-length "card images". Each card image consists of a variable-length portion holding control

Table IV

MODIFY Directory Contents

Information Needed	Type
- Number of the next REC available	Integer
- Next available directory word	Integer
- *Deck/$Comdeck/+Ident name	Character
- REC number	Integer
- Number of data cards in the deck/ comdeck/ident	Integer
- Directory entry of rest of deck/ comdeck or that deck/comdeck which ident updates	Integer

information and a variable-length portion holding a compressed version of the data record's text. The variable-length control information portion consists of all but the last element identified in Table V; the last element makes up the variable-length text portion. The length of the control information portion will vary depending on the number of times the card image changes status (active to inactive, inactive to active). Any time the status changes, one set of correction set information (CSI) elements (CSI_n#1 and CSI_n#2) will be generated, where "n" is 1, 2, 3, ..., for the first, second, third, ..., change made. A more detailed description of the deck/comdeck record format is presented later in this section.

Table V

MODIFY Card Image Contents

Information Needed	Type
- Status (1=active, 0=inactive)	Integer
- Sequence number within the deck/ comdeck/ident introducing the card	Integer
- Number of correction set information (CSI) sets to follow	Integer
- CSI_n#1 Action of correction set (1=activated card, 0=deactivated card)	Integer
- CSI_n#2 Index to array identifying the ident that introduced/ changed the card	Integer
- Number of spaces preceding the text	Integer
- Number of characters in the text	Integer
- Text	Character

The next file structure characteristic requiring consideration for the program library file was selecting coded-type or binary-type records or, in FORTRAN 77 terminology, formatted I/O or unformatted I/O. Unformatted I/O is execution-time efficient (and, usually, storage efficient) since a memory-to-storage, storage-to memory conversion is not necessary (Ref 10:342). A formatted I/O file would, on the other hand, allow for easy transport from one machine to another. However, the sequential, formatted SOURCE file should be used for this purpose.

The efficiency realized with unformatted I/O is indeed

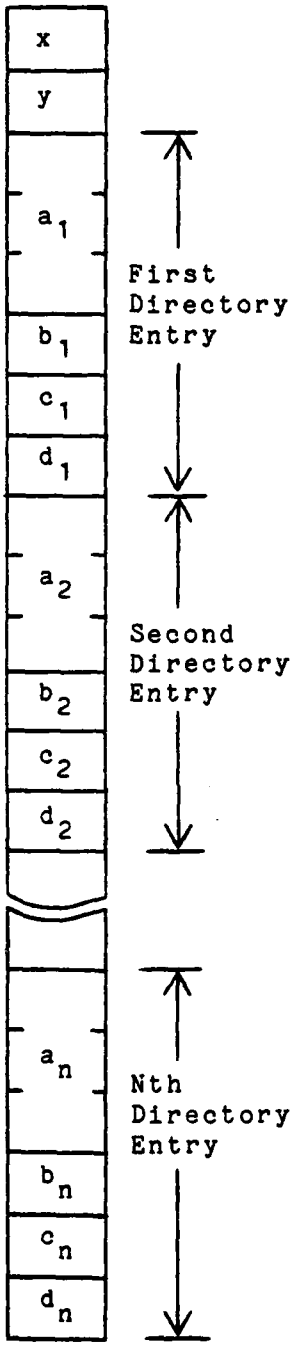
attractive; however, because of the variable-length card images in a record, the use of unformatted READ and WRITE statements, with predefined variable names, could not be used. The possibility of reading a record into an array was evaluated next; however, all data elements stored in an array are required to be of the same type (i.e., defined as CHARACTER, INTEGER, or REAL). This requirement presented a problem, as can be seen in reviewing Table IV and Table V, since the contents of the library records are made up of a mixture of CHARACTER-type and INTEGER-type data elements. FORTRAN does not allow the assignment of one data-type (e.g., CHARACTER) to a different data-type (e.g., INTEGER) within a program unit. However, since FORTRAN does not make the "consistency" check of items passed to a subroutine, the problem was eliminated. That is, a CHARACTER-type element can be passed to a subroutine and the corresponding dummy argument defined in the subroutine can be of type INTEGER. Within the subroutine, the INTEGER-defined element is moved to a second INTEGER-defined element, which is also defined as a dummy argument and used to pass the element back to the calling program. The calling program's second actual argument would be defined as INTEGER, thereby having an INTEGER-defined version of the CHARACTER-defined data element.

The final decision was made, then, to have unformatted I/O records which would be generated/retrieved via an INTEGER-type array. That is, a program library record is

written from, and read into, an integer array defined in main memory. Each integer element defined in Table IV and Table V occupies one word of memory and the character elements occupy a variable number of words in memory, depending on the number of characters in the element and the number of bytes in a word (machine dependent).

Figure 4 graphically displays the contents of REC #1, the program library directory, as it appears in the integer array. Each box represents one word in memory. Three words are reserved for the character-defined deck/comdeck/ident name. This element could have been made variable, but an additional word would have been necessary to keep track of the number of characters in the name. The number of words needed depends on the length of the character string and the number of bytes in a word. The maximum length allowed for a deck/comdeck/ident name is nine characters and with the * to designate a deck, or $ to designate a comdeck, or + to designate an ident, the element could be up to ten characters in length. The number of bytes per word on the DEC VAX 11/780 is four and the number of bytes per word on the CDC Cyber 74/750 is ten. Therefore, if the name was ten characters, three words would be needed on the VAX and only one word on the Cyber. To allow for the worst case and make it easier to "skip" through the array, three words are reserved for the deck/comdeck/ident name.

Figure 5 graphically displays the contents of each remaining record in the program library file, as they would



The fixed-length portion is represented by x and y,

where x is the number of the next available REC

y is the next available directory word

The variable-length portion is made up of a variable number of directory entries in which one entry is represented by a, b, c, and d,

where a is the *DECK/$COMDECK/+Ident name

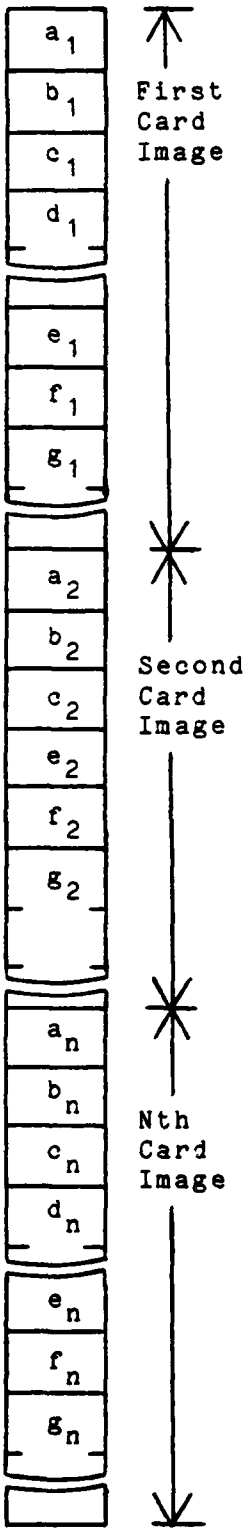
b is the (first) REC number where the contents of the deck/comdeck/ident in "a" is stored

c is the sequence number of the last input-data card introduced in the deck/comdeck/ident in "a"

d is an index to the directory entry which contains continuation information for the deck/comdeck/ident in "a"

n is the total number of directory entries generated.

Figure 4. Directory Record Contents



One card image contains a variable-length control information portion represented by a,b,c,d,e,and f, and a variable-length text portion represented by g,

where a is the status (1=active, 0=inactive) of the card image

b is the sequence number within the deck/comdeck/ident introducing the card

c is the number of correction set information (CSI) sets in "d"

d is the area of the "c" CSI sets; a set consists of two words where the first word represents the action of the correction set (1=activated card, 0=deactivated card) and the second word contains an array index identifying the ident causing the change.

e is the number of spaces preceding the text portion

f is the number of characters in the text portion

g is the text portion

n is the number of card images in one record

Figure 5. Deck/Comdeck Record Contents

appear in an integer array. A variable-length "card image" is generated for each card in a deck/comdeck. Again, each box represents one word of memory.

As card images are generated, the number of words needed depends on the way the card image is introduced, the number of characters in the text portion, and the number of bytes in a word. A card image introduced with a *DECK or *COMDECK directive does not need any correction set information (CSI) sets, represented by "d" in Figure 5; therefore, the card image would occupy five words for control information and a variable number of words for the text portion (represented by "second card image" in Figure 5). On the other hand, a card image introduced by an *IDENT directive (record inserted) would need a CSI set (two words) to identify the ident introducing the card image, thereby occupying seven words for control information and a variable number of words for the text portion. Each time a card image changes status, another set of CSI words is needed, up to a maximum of 20 CSI sets (only 20 changes allowed). In the worst case, the control information section of the card image would occupy forty-five words (twenty CSI's occupying forty words plus the other five words of control information).

The number of words needed to store the character text portion of each card image is dependent on the number of characters in the text and the number of bytes in a word (same as the deck/comdeck/ident name in the directory record). In this case, since the number of characters range

from 0 to 80 characters, one word is reserved to keep track of the number of characters in the text portion, rather than reserve a fixed number of words. The least number of words occupied by a card image would be realized if a blank card was introduced via a *DECK or *COMDECK. The card image would occupy only five words, for the minimum number of control information elements. In the worst case where the card image changed status twenty times (the maximum allowed) and the text portion contained 80 characters, the card image would occupy sixty-five words on the VAX and fifty-three words on the Cyber. The following calculations show how these totals were derived.

VAX (4 bytes per word)		Cyber (10 bytes per word)
5	Fixed, Control Information	5
40	20 CSI's, 2 words each	40
20	Text portion (80 bytes)	8
65	Total Words Occupied	53

The remaining file structure characteristic requiring consideration was the length of each record in the program library file. It is not known if MODIFY will be used (by users other than AFWAL/AAWA) to maintain many little decks and/or comdecks requiring many directory entries but a small number of card images, or a few large decks/comdecks requiring few directory entries but a large number of card images. MODIFY has been designed to be as flexible as possible to the user's needs, and the record length decision

must be made prior to implementing MODIFY on any given computer system. A more detailed discussion of this issue is presented in Chapter III.

Control Card Parameters. MODIFY supports eleven of the twenty-six UPDATE parameters, but the parameters are not specified on the job control statement used to invoke MODIFY. The parameters are, instead, listed on a "MODIFY Card" (MODIFY,p1,p2,...,pn), which must be the first card in the input file. This must be the system's designated input file (the card reader) and not an aternate input file. The eleven parameters supported by MODIFY are listed in Table VI and are all optional. However, a MODIFY card must always be submitted even though there are no parameters specified.

The "mode" parameters, F and Q, specify the order that decks are to appear on COMPILE. The Full (F) mode ignores *COMPILE directives and updates all decks in the sequence encountered on the library. The Quick (Q) mode allows *COMPILE directives to specify decks written to COMPILE in the order encountered on the library, unless "K" is also specified then decks are in the order specified on *COMPILE directives. If neither Full nor Quick is specified, a Normal mode is implied. The Normal mode writes to COMPILE all decks specified on *COMPILE directives plus decks updated, in the order encountered on the library. If "K" is specified, decks are in the order specified on *COMPILE directives followed by any others updated.

The List (L) parameter allows the user to specify out-

Table VI

MODIFY Job Control Parameters

Option	Significance
C	Compile file output
F	Full update mode
I	Input file
K	Decks written to COMPILE in order specified by *COMPILE directives
L	Output listing options
N	New program library output file
O	Output listing file
P	Old program library file
Q	Quick update mode
S	Source output file
*	Directive card master control character

put listing options to be invoked. Table VII lists the acceptable values and a brief description of what each value will generate. The default value for a creation run is A12, for a correction run is A1234. All but two of UPDATE's options are supported and one option is included that is not provided by UPDATE. The UPDATE options 5 and 6 are not supported, but the numbers have been reserved so that they could easily be included in a later version of MODIFY. A new Option B for alphabetized lists was included to allow the user to easily locate a specified deck/comdeck/ident

Table VII

MODIFY List Options

Option	Significance
A	Lists known decks/comdecks/ident names and deck names written to COMPILE file
F	All selections other than 0
0	Suppresses all MODIFY listings
1	Lists cards in error and the associated error messages
2	Lists all active MODIFY directives encountered either on input or on the old program library
3	Lists all cards that changed status during MODIFY run
4	Lists all non-MODIFY directives encountered in the input stream
7	Lists all active cards
8	Lists all inactive cards
9	Correction history of all cards listed as a result of list options 7 and 8
B	Three alphabetic lists by deck, comdeck, and ident names

name. As the list of names grows, it sometimes is hard to locate specific names.

The descriptions of the remaining MODIFY control card parameters listed in Table VI are self-explanatory and are not discussed further here. A detailed description of the MODIFY Control Card Parameters, along with the MODIFY directives, can be found in the MODIFY User's Guide, Appendix C.

III. Development of the Program

Writing a machine-independent software package cannot be accomplished without requiring some machine-dependent information. The first section of this chapter defines the information which must be supplied before MODIFY is implemented. The second section outlines in detail how directory entries are generated which, in turn, can decrease the expected number of deck/comdeck/ident names allowed. The last section gives an overview of how the program works.

Machine-Dependent Requirements

This section defines the information which must be supplied prior to implementing MODIFY. The information required consists of the values needed for three constants, an integer array, and a character array.

Default File Names. The FORTRAN 77 OPEN statement contains an optional parameter, FILE, which gives the name of the file being connected to the unit. This parameter is used in the MODIFY OPEN statements; therefore, default names for the six user-accessible files must be assigned. The values assigned are placed in a character array.

The assignment of these file names could have been fixed; however, the assignment has been left until compilation so that file names corresponding to each particular system could be used. The decision to use the FILE parameter will also allow the user unfamiliar with

FORTRAN default names to use MODIFY without having to learn those names (e.g., the VAX default is FOR0xx, and the Cyber default is TAPExx, where "xx" is the unit number). It is recommended that the system-default names be used for the input and output files (i.e., VAX input is SYS$INPUT and output is SYS$OUTPUT; Cyber input is INPUT and output is OUTPUT). It is also recommended that standard names be selected for the remaining user-accessible files to avoid any confusion when working with different computer systems. That is, using the names OLDPL, COMPILE, NEWPL and SOURCE as default file names for the remaining user-accessible files should eliminate any ambiguity about the contents and/or purpose of the files. The use of these file names would also correspond to the names used throughout this report, as well as the MODIFY User's Guide, and would also help in eliminating any ambiguity.

The last entry of the character array defines the default value of the directive card master control character. It is recommended that the asterisk be assigned to correspond with the user's manual. However, the programming language used by the majority of MODIFY users may dictate that a character other than an asterisk be used. For example, standard FORTRAN 77 permits the use of an asterisk in column 1 to denote a comment. If the asterisk is used extensively to denote a comment, it may be desirable to use a different master control character. An asterisk as the default value would force users to change the default

value every time MODIFY is run.

File Name Length. The maximum length of the file names must be provided. At compilation time the default names are provided, but the user of MODIFY still has the option of changing the default names (via the MODIFY control card parameters). A check is made to insure that the name submitted on the MODIFY card does not exceed the system's filename limit (e.g., VAX = 9 characters, Cyber = 7 characters).

Unit Numbers. FORTRAN requires that all READ and WRITE statements identify a unit number (1-99) corresponding to the appropriate file. Some computer systems assign a unit number to a specified device (e.g., the VAX assigns unit 5 to the card reader and unit 6 to the printer). The eleven files used by MODIFY, therefore, must be assigned a unit number. The values are placed in an integer array and used accordingly. It is recommended that system default unit numbers be used when appropriate (e.g., on the VAX), otherwise any eleven numbers from 1 to 99 may be selected.

Word Length. The number of bytes in a word makes a significant difference in determining the size of the program library file. This information varies from one computer system to another and must be provided. The difference between the VAX and Cyber illustrates this variability. The VAX is a 32-bit machine, with 8-bit bytes, and 4-byte words. The Cyber is a 60-bit machine, with 6-bit bytes, and 10-byte words. The reason for needing this

information has already be explained in Chapter II's "MODIFY Characteristics" section.

Record Length. The remaining constant required deals with the length of each record in the program library file. FORTRAN requires that this assignment be made for direct-access files, and the unit of assignment is machine-dependent.

The record length value selected has a tremendous impact on storage efficiency and places some limits as to the capabilities of MODIFY. Because of the importance of selecting an appropriate value, a separate section has been written to discuss the impact of this assignment.

Processing Limitations

This section outlines the importance of selecting an appropriate value for the record length parameter, RECL, which appears in the OLDPL and NEWPL OPEN statements. Since both the VAX and Cyber requires the unit of assignment to be in "words", this discussion will focus on determining the number of words in each record.

The contents of the directory record consists of two one-word elements and a variable number of six-word elements (a directory entry). Recall that at least one directory entry is generated for each deck/comdeck/ident introduced (see Figure 4). Therefore, a record length of 602 words would allow for 100 deck/comdeck/ident names. On the other hand, the record length of 602 words would, on the average,

allow for 40 card images on the VAX (66 on the Cyber) in a deck/comdeck record. These averages are representative of introducing a new deck/comdeck (no CSIs generated) and the average length of the text portion is 40 characters. The following calculations show how the averages were derived.

VAX (4-byte words)		Cyber (10-byte words)
5	Control Information (no CSI words)	5
10	Text portion (40 bytes)	4
15	Words per card image	9
	Divide the values into 602 words	
40	Card images per record	66

The example shows that a record would hold only a minimal number of card images (card-input records) and does not account for the need of additional words when updates are requested. Additional words are needed each time a card image is changed (a 2-word CSI is generated) and anytime a new card-input record is introduced (a new card image is generated). On the other hand, if a large number of small comdecks (say 2-to-3 lines of code) are introduced, the 40 (or 66) card images would be more than enough, and space would be wasted.

In an effort to eliminate wasted space but allow a deck/comdeck to grow, MODIFY will automatically place card images in continuation records and update the directory accordingly. This is all transparent to the user, but does

require an additional entry in the directory for each continuation generated, decreasing the number of decks allowed in the program library.

Multiple directory entries may also be needed for any one ident name introduced. An ident name is used to change and/or add card images to already existing decks/comdecks. If the directives in one run requests changes to card images in, say, five different decks, then five directory entries will be generated. Each entry will identify the program library record updated. If a change to any of the ident card images is then requested, MODIFY needs only search those five records to find the card image, and not all the records in the library.

Because of the continuation of decks/comdecks and multiple entries needed for idents, the directory must be larger than just allowing for a reasonable number of decks/comdecks/idents. Once the directory is full, MODIFY will no longer process any update-type requests. There is a way, however, to allow changes to continue. At the end of each run MODIFY will automatically print the number of entries left in the directory. There is no provision to "clean up" the directory so, when the entries are all used, it is suggested that a SOURCE file be generated. The SOURCE file can then be designated as the input file to a MODIFY creation run. This will eliminate all the historical information, but will allow additional changes to be made.

It is recommended that the record length assignment be

made based on the average size of decks/comdecks (to save space) but also allow the directory to have room for twice as many decks/comdecks/idents expected. See Appendix B for a discussion of the record length selected for the DEC VAX 11/780 and CDC Cyber 74/750 computer systems.

Program Description

The MODIFY program is used to maintain and retrieve source files located in a program library. This concept allows the user to store multiple source files under one file name thereby using less computer resources (e.g., one entry in the file name table). MODIFY also allows the user to define "common code" which requires only one change to the common deck rather than a change for every place the code is used. The main advantage of MODIFY is, however, the capability of tracking all previous changes made to the source code. By revoking the effects of an ident (via *YANK), the source code changed is returned to its previous state. All operations of the MODIFY program occur in response to the user-supplied MODIFY Card and the input transaction file containing MODIFY directive cards and/or data cards.

The MODIFY Card must be the first card on the input transaction file and may optionally include up to eleven parameters (see Table VI). The MODIFY parameters allow the user to specify which files are to be generated, the run mode, and the order in which the decks are to be processed.

The fourteen directives (see Table II) allow the user to specify the exact line(s) of code to be updated and/or the decks to be retrieved for compilation. A more detailed description of the parameters and directives can be found in the MODIFY User's Guide, Appendix C.

Program Structure. The basic functional structure of the MODIFY program is indicated in Figure 6. Appendix D provides a more detailed set of Structured Analysis and Design Technique (SADT) charts which depict the primary configuration of the program. MODIFY consists of the main program and 54 subroutines. A brief description of each subroutine is provided in Appendix E. The main program basically contains six CALL statements which causes each of the four primary functions to be executed (SADT charts do not depict housekeeping functions).

The "Set Control Information" routine processes the MODIFY Card. The card is validated and then used to set appropriate values in three arrays. A character array contains the file names and the directive card master control character. One logical array is used to indicate which files are to be generated, the run mode (full, quick, or normal), and the run type (creation or correction). Another logical array is used to indicate which list options are to be generated. The information contained in these arrays is used continually by the update routines.

The "Set Processing Information" routine processes the input transaction file. Each directive is validated and

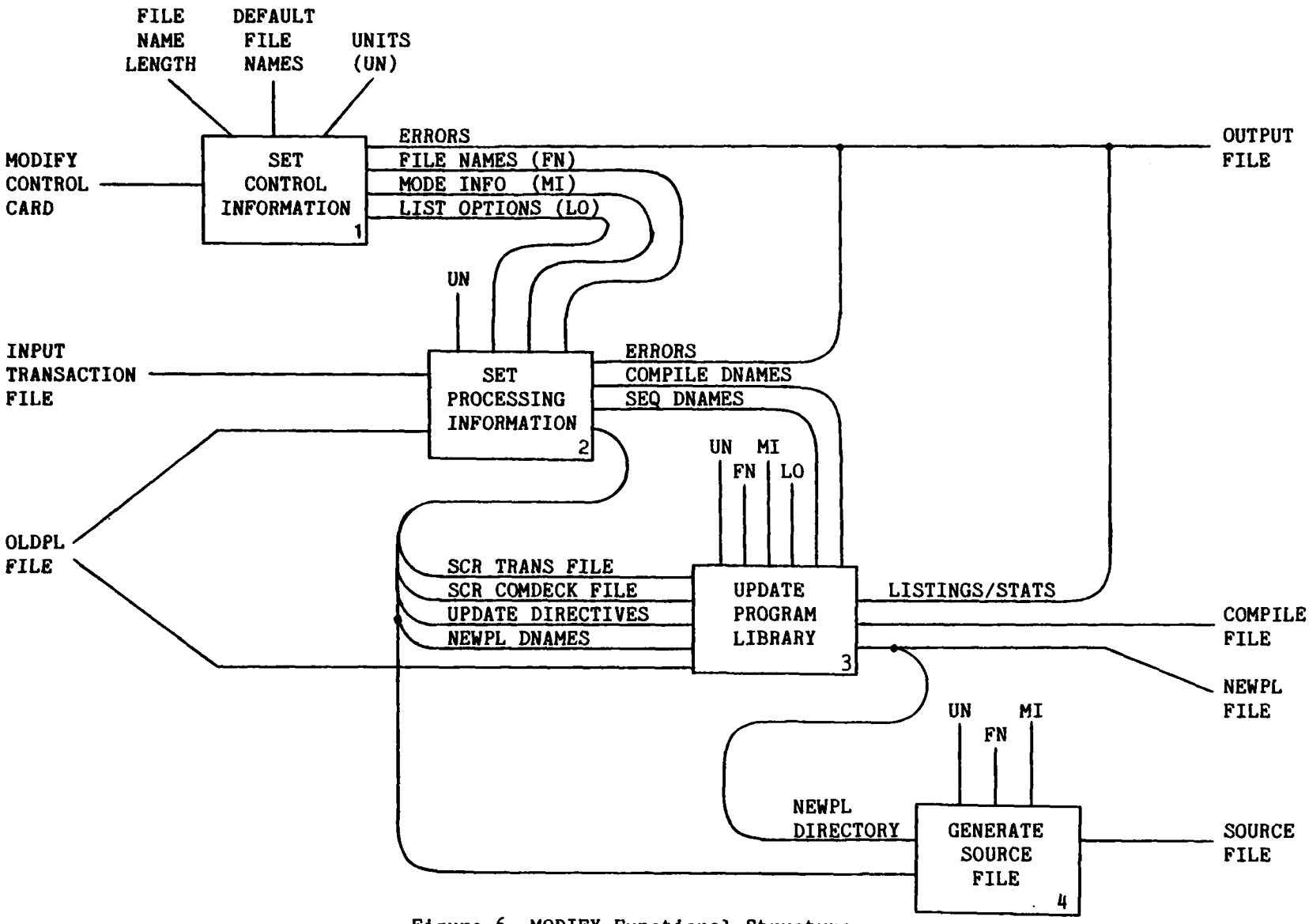


Figure 6. MODIFY Functional Structure

selected information is retrieved and placed in the appropriate arrays. Those directives which can have data cards following (the *DECK, *COMDECK, *DELETE, *INSERT, and *RESTORE) are placed in the scratch input file or the scratch comdeck file, along with the data cards. The information generated by this routine is also used continually by the update routines.

The "Update Program Library" routine controls the bulk of the work. The control information depicting the user's desires is all available and the actual updating/retrieving begins. The creation run requires little processing since no updating takes place. Each record on the scratch comdeck file and the scratch input file is used to generate a card image for the NEWPL file and the text portion is written to the OUPUT file. The text portion of non-directive card images is written to file COMPILE, replacing *CALL directives with the contents of the specified comdeck.

The correction run tests the overall program strategy; that is, to locate a particular line of code as efficiently as possible, make the specified change, and provide the user with an updated copy suitable for compilation. The old program library directory is brought into memory, and selected data is merged along with the new deck/comdeck/ident data (generated from the input transaction file by the "Set Processing Information" routine) into another array. The deck/comdeck/ident names in the merged-data array are in the order of (1) previous ident names, (2) new ident names,

and (3) the deck and comdeck names. New deck/comdeck names are placed after previous deck/comdeck names unless an *ADDFILE directive identifies a specific order. If an *ADDFILE does identify a specific order, the new deck/comdeck names will be inserted appropriately. The order of the deck names can be important because this merged-data array is used to determine the order that decks are processed (order of decks in the COMPILE file).

Any directives submitted with references to the contents of a comdeck are processed first. The appropriate program library record is brought into memory and the specified change is made. The text portion of each active card image is written to the scratch comdeck file and the text portion of all card images is written to the OUTPUT and NEWPL files (according to MODIFY control card parameters selected). After all selected comdecks have been updated or introduced, any remaining comdeck records are brought into memory and the text portion of the active card images is written to the scratch comdeck file. The scratch comdeck file is used when the contents of decks are written to COMPILE (to replace *CALL directives with the contents of the comdeck specified on the *CALL).

Any remaining directives, referencing the contents of a deck, are processed next. The order of the decks selected depends on the run mode. The "full" mode selects decks in the order encountered on the merged-data array. The "quick" mode selects those decks specified on *COMPILE directives,

in the order encountered on the merged-data array. However, if the "K" control card parameter is specified, the decks are selected in the order specified on the *COMPILE directives. The "normal" mode selects those decks specified on *COMPILE directives and decks to be updated, in the order encountered on the merged-data array. However, if the "K" control card parameter is specified, the decks are selected in the order specified on the *COMPILE directives followed by the decks updated in the order encountered on the merged-data array.

As new deck names are selected, the scratch input file is searched looking for the deck's input-data records. Each input data record is used to generate a card image, and the text portion of the card image is written to the appropriate files. Continuation records are generated automatically as they become necessary.

As previously known deck names are selected, the REC number element of the directory entry is used to bring the deck's (first) record into memory. If the deck is not to be updated (no directives reference the contents) the record is written to NEWPL. If the deck is to be updated, the text portion of each card image is brought out of its compressed format (regains its original image in a 80-byte field).

The card image sequence number is examined, looking for a match with the sequence number specified on the deck's update directives. If the sequence numbers do not match, the text image is written to the OUTPUT, COMPILE, and/or

NEWPL files. If the sequence numbers match and a change is to be made, the status of the card image is changed (active to inactive or inactive to active), the text image is written to OUTPUT, and a card image is generated for the NEWPL record. The specified change is made to the text image, the new text image is written to the OUTPUT and COMPILE files and a card image is generated for NEWPL. If more card images are located in a continuation record (indicated by the fourth element in the directory entry), that record is "brought in" and the process continues until all records storing the deck's card images have been processed.

A certain amount of processing control is handled by the routines used in building the OUTPUT, COMPILE, and NEWPL files. A carriage control character is assigned to each OUTPUT record written, allowing only a predefined number of printed lines per page, starting each page with a header and numbering each printed page. The routine that builds COMPILE must examine each active card image and replace any *CALL directives with the appropriate comdeck records. The routine that builds NEWPL insures that there are enough words available in the integer array to hold the card image. If the card image will not fit, the array is written to the appropriate NEWPL record and a continuation record is started. If the previous version (OLDPL) did not require a continuation record for the deck, the next available REC number is fetched, the directory entry updated, and a new

directory entry generated.

Locating all the lines of code specified on a *YANK directive can require more search and processing time. The *YANK identifies correction sets to be revoked, where a correction set may have changed card images spanning across a variable number of decks/comdecks, which may span across a variable number of records. The ident name is located in the directory and the REC number element of the entry is used to bring into memory the first deck containing a change specified by the correction set. All the card images in the deck's record(s) are examined and written to the appropriate files. If the ident made changes to, say, five different deck names and the contents of each deck is stored in two records, the card images in the ten records would be examined and processed as previously explained.

Once all decks have been processed, the NEWPL directory record is generated. The OLDPL directory and the merged-data array are used to generate the NEWPL directory.

The "Generate Source File" routine is executed only if the user requests that the SOURCE file be generated. The routine uses the NEWPL directory to first retrieve comdeck records and then deck records, writing only "active" records to SOURCE.

Data Structures. The program contains common blocks to allow the subroutines easy access to the common data structures. The common area includes 28 integer elements, 10 integer arrays, 6 character elements, 12 character

arrays, and 5 logical arrays. One of the lower-level subroutines uses a stack structure to control the building of file COMPILE.

The scratch comdeck file is used to contain the records of all common decks, new and old. When file COMPILE is to be generated with the contents of requested deck records, each *CALL directive is replaced with the contents of the comdeck specified. As comdeck records are written to file COMPILE, any *CALL directives must also be replaced with the contents of that comdeck specified. The stack structure is used to store the placement of a *CALL directive in the scratch comdeck file so that remaining records of a comdeck will also be written to COMPILE.

This concludes the discussion of analyzing and developing the machine-independent utility MODIFY. Version 1 was implemented on the DEC VAX 11/780.

IV. Conclusions

The primary objective of this project, as outlined by AFWAL/AAWA requirements, was to provide a machine-independent software package emulating a subset of CDC's UPDATE utility program. An additional objective was to include additional features not available on UPDATE, specifically a string search and replacement option. This chapter summarizes the results of the MODIFY design and implementation effort and provides a few recommendations to enhance MODIFY.

Results

The MODIFY specifications described in the preceding chapters of this report were completely designed, coded, and implemented. However, as of this date, not all of the functions have been thoroughly tested and validated. The complete package should be available by the end of this month, accomplishing all the functions described.

Recommendations

Three areas of improvement have been identified and will be discussed: additions to the MODIFY command language, revisions of the program to provide interactive capabilities, and the inclusion of more output list options.

MODIFY is designed to accomplish normal update/ retrieval functions; however, as indicated by the number of

commands available in CDC's UPDATE package (e.g. 41 directives), MODIFY is limited (14 directives). The inclusion of additional directives would enhance MODIFY, specifically in the area of removing obsolete names from the program library directory.

MODIFY is designed to accept batch inputs (via cards or intercom). Because of the growing availability of CRTs, MODIFY would be a more valuable tool if it was redesigned to allow for interactive use. That is, include the logic necessary to prompt the user for the actions desired.

The last recommendation to enhance MODIFY includes the addition of more output list options. The two options included in UPDATE but not in MODIFY, are suggested: A listing of all active compile file directives and a listing giving the number of active/inactive cards by deck name and correction set identifier.

Bibliography

1. Applied Data Research. The LIBRARIAN User Reference Manual, SL2G-10-00. Release 5.5. Applied Data Research, Inc., August 1976.

2. Balfour, A. and D. H. Marwick. Programming in Standard FORTRAN 77. New york: North-Holland Inc., 1979.

3. Control Data Corporation. Terminal Facility Controller EDIT Program Reference Manual, 82173400. St. Paul, Minnesota: Control Data Corporation, 1973.

4. Control Data Corporation. UPDATE Reference Manual, 60342500. Revision H. Sunnyvale, California: Control Data Corporation, January 1978.

5. Graham, David C., Lewis P. Gaby, II, and Clifford E. Rhoades, Jr. SAIL, An Automated Approach to Software Development and Management. Kirtland Air Force Base, New Mexico: Air Force Weapons Laboratory, Air Force Systems Command, October 1976.

6. Hurt, James J. MEDIT--A Program to Edit Computer Source Programs. DTIC AD Number: A034524. U. S. Army Armament Command, Research Directorate, April 1976.

7. Kessinger, Richard L. and Howard A. Markham. Models of the US Army Worldwide Logistic System (MAWLOGS), Volume IV, Programmers's Guide. Contract DAHC19-69-C-0017; DTIC AD Number: 923126L. McLean, Virginia: General Research Corporation, August 1974.

8. OpCode. The HISTORIAN User's Manual. Houston, Texas: OpCode, Inc.

9. Pansophic Systems. PANVALET User Reference Manual OS, OSUP10-7811. Version 10. Pansophic Systems, Inc.,1978.

10. Wagener, Jerrold L. FORTRAN 77 Principles of Programming. New York: John Wiley & Sons, 1980.

Appendix A

UPDATE's Capabilities and Files

Appendix A

UPDATE's Capabilities and Files

This appendix contains five tables providing a detailed account of the features, directives, file characteristics, job control parameters, and list options of the Control Data Corporation (CDC) UPDATE utility package. A discussion of these items may be found in the "UPDATE Characteristics" section of Chapter II.

Table A-I

UPDATE Features

- Creation of a program library from source decks.
- Copying of old program libraries from sequential to random format and vice versa.
- Merging of two program library files.
- Updating of source decks by inserting, deleting, and restoring cards according to sequence in the deck or according to correction set.
- Ability to completely and permanently remove correction sets from the program library.
- Generation of a compile file containing corrected output acceptable as input to other processing programs, such as compilers and assemblers.
- Processing of directives, new text, and new source decks from a file other than the job input file.
- Production of fresh source decks from the program library.
- Generation of a new, updated program library.
- Comprehensive list output noting any changes occurring during the run and status of the program library.
- Ability to change the directive card master control character.
- Recognition of abbreviated forms of directives and capability of turning off the search for the abbreviated forms to speed up processing.
- Ability to use full 64-character set, including the colon.
- Checksumming of program library.

Table A-II

UPDATE Directives

Name	Abbreviation	Function
*ABBREV	none	Abbreviations will be recognized
*ADDFILE	*AF	Add a deck or comdeck to the PL
*BEFORE	*B	Adds the records following the BEFORE before specified location
*CALL	*CA	Replaces this directive with the contents of the comdeck identified when the deck/comdeck is written to file COMPILE
*CHANGE	*CH	Change the name of a correction set defined by an IDENT
*COMDECK	*CD	Introduces a common deck, common code which may be inserted into a deck or another comdeck when CALLed
*COMPILE	*C	Specifies decks to be written to file COMPILE for later compilation
*COPY	*CY	Copy specified records from one deck/comdeck to another deck/comdeck
*CWEOR	*CW	Conditionally writes an end-of-file indicator in file COMPILE
*DECK	*DK	Introduces a source deck, usually contains a source program but can contain job control language, sub-programs, data, etc.
*DECLARE	*DC	Identifies decks/comdecks to be altered, protects against inadvertant updates
*DEFINE	*DF	Establishes conditional condition to be tested by *IF
*DELETE	*D	Specified records are flagged as inactive and optionally replaced with records following the DELETE
*DO	none	Reverses the effects of YANK or SELYANK on specified correction set

Table A-II, Con't

UPDATE Directives

Name	Abbreviation	Function
*DONT	*DT	Terminates *DO
*END	none	Denotes end of source code
*ENDIF	*EI	Indicates the end of conditional text, terminates *IF
*ENDTEXT	*ET	Indicates the end of text, terminates *TEXT
*IF	none	Conditionally writes text to file COMPILE
*IDENT	*ID	Introduces a correction set
*INSERT	*I	Adds the records following the INSERT after the specified location
*LIMIT	*LT	Specifies maximum size for the list output file
*LIST	*L	Terminates effects of *NOLIST
*MOVE	*M	Reorders decks while producing NEWPL
*NOABBREV	*NA	Do not accept directives abbreviation format
*NOLIST	*NL	Stops the listing of input file records
*PULLMOD	*PM	Recreates correction sets
*PURDECK	*PD	Permanently removes all records of specified deck/comdeck
*PURGE	*P	Permanently removes all records of specified correct set, restores decks/comdecks to previous state
*READ	*RD	Temporarily read from an alternate directives file then return to INPUT
*RESTORE	*R	Specified records are reactivated and optionally inserts the records following RESTORE

Table A-II, Con't

UPDATE Directives

Name	Abbreviation	Function
*REWIND	*RW	Repositions specified file to first logical record
*SELPURGE	*SP	Permanently removes records of specified correction set located in specified deck/comdeck
*SELYANK	*SY	Deactivates records of a correction set located in specified deck/ comdeck
*SEQUENCE	*S	Resequences active cards and purges inactive cards in specified deck/comdeck
*SKIP	*SK	Skip specified logical records in specified file
*TEXT	*T	Treats following records as data, allowing data to have asterisk in column 1
*WEOR	*W	Writes end-of-file indicator in file COMPILE
*YANK	*Y	Deactivates all records of specified correction set, restores decks/comdecks to previous state
*YANKDECK	*YD	Deactivates all records of specified deck/comdeck
*/	none	Indicates comment card

Table A-III

UPDATE File Characteristics

File Name	Type	Storage	Function
INPUT	Coded	Sequential	Provides control information
OUTPUT	Coded	Sequential	Contains LIstings
COMPILE	Coded	Sequential	Contains card images for compilation
SOURCE	Coded	Sequential	Contains "active card images with no sequencing information
OLDPL	Binary	Random or Sequential, Compressed	Contains old program library
NEWPL	Binary	Random or Sequential, Compressed	Contains new program library
UPDTSCR	**	**	Used to make copy of decks to be written to file COMPILE
UPDTCDK	**	**	Used to hold common decks for later expansion of *CALL cards
UPDTTPL	**	**	Used as temporary program library
UPDTEXT	**	**	Used to copy card images to be inserted in correction run
UPDTAUD	**	**	Used to hold temporary audit information
UPDTPMD	**	**	Used to collect card images in response to PULLMOD directive

** Information not provided

Table A-IV

UPDATE Job Control Parameters

Option	Significance
A	Sequential-to-random copy
B	Random-to-sequential copy
C	Compile file output
D	Data width
E	Edit; provides a means of cleaning up old program libraries
F	Full update
G	Generate separate PULLMOD output file
H	Header change
I	Input file
K	COMPILE file sequence
L	List options
M	Merge two program libraries
N	New program library output
O	List output file
P	Old program library
Q	Quick update
R	Rewind files
S	Source output file
T	Source output file excluding common decks
U	Debug mode
W	Sequential new program library
X	Compressed compile file

Table A-IV, Con't

UPDATE Job Control Parameters

Option	Significance
Z	Compressed input file
8	80-column output on compile file
*	Master control character
/	Comment control character

Table A-V

UPDATE List Options

Option	Significance
A	Lists known decks/comdecks/ident names, deck names written to COMPILE file, and known definitions
F	All selections other than 0
0	Suppresses all UPDATE listings
1	Lists cards in error and the associated error messages
2	Lists all active UPDATE directives encountered either on input or on the old program library
3	Lists all cards that changed status during UPDATE run
4	Lists all non-UPDATE directives encountered in the input stream
5	Lists all active compile file directives
6	Lists the number of active and inactive cards by deck name and correction set identifier
7	Lists all active cards
8	Lists all inactive cards
9	Correction history of all cards listed as a result of list options 5, 7, and 8

Appendix B

VAX/Cyber Record Length Assignment

Appendix B

VAX/Cyber Record Length Assignment

FORTRAN 77 requires that the length of records in a direct-access file be provided by the RECL parameter on the OPEN statement. Because the MODIFY program library is a direct-access file, the record length assignment must be determined prior to compilation. This appendix provides a detailed discussion of the record length values selected for the DEC VAX 11/780 and the CDC Cyber 74/750 computer systems, as required by the AFWAL/AAWA processing environment. The unit of assignment is machine dependent, but since both the VAX FORTRAN and CDC FORTRAN requires that the assignment be in "words", this discussion will focus on the number of words needed.

The AAWA personnel have been using program libraries to store a large number of decks containing a minimal number of cards. The requirement was established by AAWA personnel that a record length value be selected which would accommodate approximately 500 directory entries and allow each program library record to contain approximately 150 card images.

VAX Record Length Assignment

The smallest addressable unit of information on a disk is a 512-byte block (128 words). Blocks are logically grouped into a cluster which is the basic unit of disk space

allocation. The number of blocks in a cluster is installation dependent and currently AAWA personnel have each of their three disk packs configured differently (3, 6, and 11 blocks in a cluster). The disk pack to be used to store MODIFY contains 3 blocks in a cluster; therefore, the record length value selected must be in increments of 384 words (evenly divisible by 384):

3 blocks * 512 bytes/block ÷ 4 bytes/word

= 384 words/cluster

The requirement to allow 500 directory entries requires that the record length be at least 3002 words:

((500 entries * 6 words/entry) + 2 words/control)

= 3002 words/record

To allow 500 directory entries and to stay within a 3-block cluster, the record length assignment of 3072 words was selected:

3002 words/record ÷ 384 words/cluster

= 7.8 clusters/record

384 words/cluster * 8 clusters

= 3072 words/record

The 3072-word record length accommodates 511 directory entries. Two words are used for the directory control information (the next REC number available and the next directory word available), leaving 4 words unused:

3072 words/record ÷ 6 words/entry

= 512 six-word entries

512 six-word entries - 1 six-word entry

= 511 directory entries

1 six-word entry - 2 words for control

= 4 words unused

The control information portion of each card image requires an average of 7 words (this includes one set of correction set information words). The text portion of each card image requires an average of 8 words (based on 32 bytes of text ÷ 4 bytes/word). Therefore, each card image requires an average of 15 words. The 3072-word record length accommodates approximately 200 card images:

3072 words/record ÷ 15 words/card image

= 204.8 card images/record

The allowance of 200 card images is above the 150-card-images-per-record requirement; however, many changes to the contents of decks can be expected. The AFWAL/AAWA

environment involves "testing an idea" which requires many correction sets (many ident directory entries). Each change in the correction set, in turn, causes a 2-word correction-set-information set to be generated for each card image specified in the change. The 3072-word record reasonably accommodates both the 500-directory-entries-per-record requirement and the 150-card-images-per-record requirement.

Cyber Record Length Assignment

The smallest addressable unit of information on a disk is a block, where a block must begin on a word boundary. The requirement to allow 500 directory entries requires that the record length be 3002 words:

((500 entries * 6 words/entry) + 2 words/control)
= 3002 words/record

where the "2 words/control" is reserved for the next REC number available and the next directory word available.

The control information portion of each card image requires an average of 7 words (this includes one set of correction set information words). The text portion of each card image requires an average of 4 words (based on 40 bytes of text + 10 bytes/word). Therefore, each card image requires an average of 11 words. The 3002-word record length accommodates approximately 275 card images:

3002 words/record ÷ 11 words/card image

= 272.91 card images/record

The allowance of 275 card images is well above the 150-card-images-per-record requirement. As previously stated, the AFWAL/AAWA processing environment involves a lot of testing of ideas, where changes are made to see the effects. The 3002-word record length assignment may cause a lot of wasted space in each record (depending on the number of card images in the record and the number of correction set information words generated), but the assignment will accommodate the directory-entry requirement.

This discussion shows that the record length assignment is dependent upon the user's needs. Depending on the processing environment, a trade-off may have to be made; that is, allow for many correction sets in the directory and waste disk space in the deck records or allow for fewer correction sets and allow more efficient use of the disk space.

Appendix C

MODIFY User's Guide

Appendix C

MODIFY User's Guide

Table of Contents

Appendix C

MODIFY User's Guide

I. Introduction

MODIFY is a machine-independent software package written in standard FORTRAN 77 and is capable of updating and/or retrieving sets of data stored in one file called a program library (PL). A program library contains "decks", "comdecks", and "idents". A "deck" usually consists of a source program, but may contain job control language, subprograms, data, etc. A "comdeck", or common deck, contains common source code which may be inserted into multiple decks or other comdecks. An "ident" identifies a run correction set which includes directives and, optionally, source-code records used in updating a deck or comdeck.

Writing a machine-independent software package cannot be accomplished without some machine-dependent information. Table C-I lists the required information which must be provided before MODIFY is compiled, along with the values assigned for the DEC VAX 11/780 and CDC Cyber 74/750 computer systems.

Chapter II provides a brief overview of MODIFY while Chapter III a provides more detailed description of how the user may use MODIFY. Chapter IV outlines in detail the required user-supplied MODIFY Control Card, and Chapter V outlines the optional user-supplied directive cards. Chapter VI provides examples of job control streams for both the VAX and Cyber computer systems.

Table C-I
Machine-Dependent Information

Information Needed	Value Supplied	
	VAX	Cyber
Word Length (bytes per word)	4	10
Record Length (words per record)	3072	3002
File Name Length (maximum allowed)	9	7
Input File Default Name	SYS$INPUT	INPUT
Output File Default Name	SYS$OUTPUT	OUTPUT
Old PL File Default Name	OLDPL	OLDPL
New PL File Default Name	NEWPL	NEWPL
Compile File Default Name	COMPILE	COMPILE
Source File Default Name	SOURCE	SOURCE
Master Control Character	*	*
Input File Unit Number	5	5
Output File Unit Number	6	6
Old PL Unit Number	1	1
New PL Unit Number	2	2
Compile File Unit Number	3	3
Source File Unit Number	4	4
Alternate Input Unit Number	7	7
Alternate Output Unit Number	8	8
Scratch Input Unit Number	9	9
Scratch Output Unit Number	10	10
Scratch Comdeck Unit Number	11	11

II. Design

This chapter provides an overview of the program library structure and a description of the various files accessible to the user.

Library Structure

The MODIFY program library is a direct-access organized file containing unformatted I/O "records". One record is used as the library's directory, in which at least one entry is generated for each deck, comdeck, and ident introduced. The remaining library records are used to store the contents of each deck/comdeck introduced. If the contents of a deck will not fit into one record, a continuation record will automatically be used and an additional entry is generated in the directory for each continuation record. An ident correction set will cause "n" directory entries to be generated, where "n" is the number of decks/comdecks updated by the ident.

One or more records are used to store "card images" for each deck and comdeck introduced. "Card images" contain control information and the text portion of each data-input card identified within a deck or comdeck. When a library record is full and more data-input cards still exist for a given deck or comdeck, MODIFY will automatically start another record and continue generating card images until the deck/comdeck data-input cards have been processed.

As data-input cards are received and card images generated, the control information portion of a card image is assigned an unique "dname.seqnbr" identifier. The identifier assigned each card image will appear with each data-input card (text portion of card image) in the output listing and can be used to request changes to the text portion of the designated card image (via MODIFY directives). MODIFY also has a "historical" feature allowing any set of changes made to the contents of decks or comdecks to be revoked. The control information portion of a card image keeps track of any ident names which changed the status of the card image. By including the ident name on the *YANK directive, the correction set is revoked.

MODIFY Files

This section provides a description of all the files accessible to the user, whereas the first section only described the program library file. MODIFY also interacts with five scratch files, but they are not described here because the interaction is transparent to the user.

MODIFY optionally interacts with six user-accessible files. Figure C-1 depicts this relationship. The dotted lines indicate optional files, the use of which is determined by the type of run (creation or correction) and the user-supplied MODIFY Control Card.

The default file names used for the VAX implementation of MODIFY appear in Table C-1 of this user's guide. The

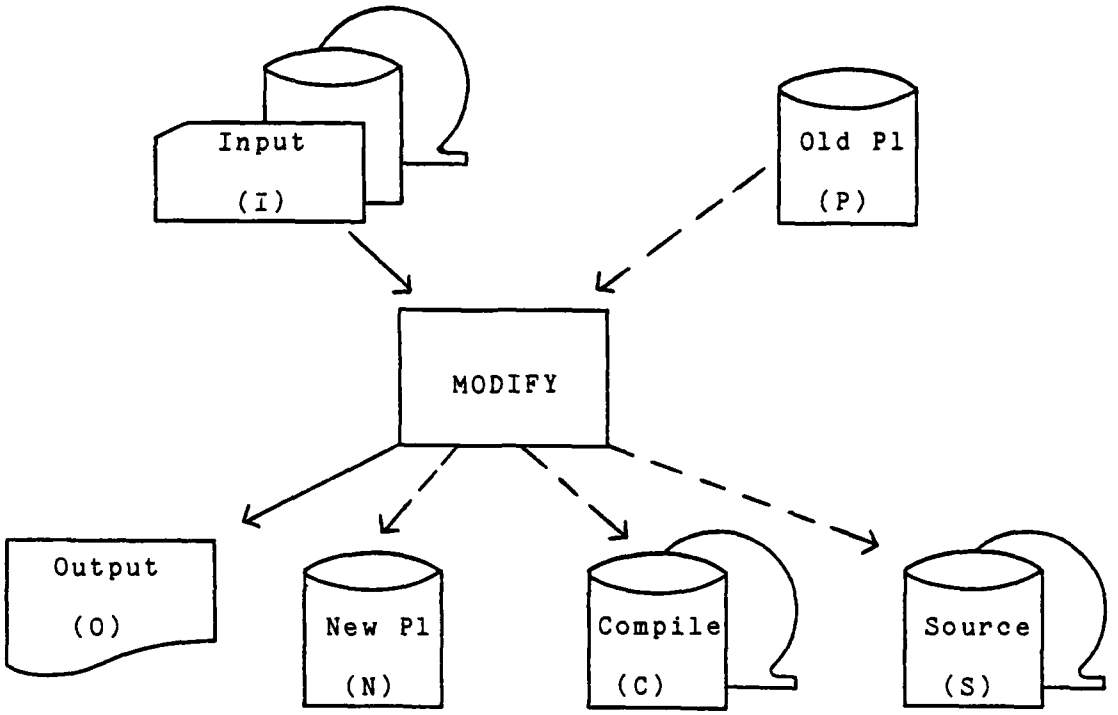


Figure C-1. MODIFY User-Accessible File Interaction

alphabetic character in parenthesis appearing in Figure C-1 indicates the MODIFY control card parameter which may be used to change the default file name. With the exception of a creation run, the old program library file is the latest version of the new program library file generated from a previous MODIFY run.

The characteristics of the MODIFY user-accessible files are listed in Table C-II. The input file must contain the MODIFY Control Card and may contain a combination of directives, or directives and data-input cards. The contents of the output file is determined by the type of run

Table C-II

MODIFY User-Accessible File Characteristics

File	Type	Storage	Record Length	Function
Old PL	Unformatted	Direct, Compressed	User-Supplied	Contains old PL
New PL	Unformatted	Direct, Compressed	User-Supplied	Contains new PL
Input	Formatted	Sequential	80	Contains MODIFY card and can contain directives and data records
Output	Formatted	Sequential	132	Contains listings requested via the "L" control card parameter
Compile	Formatted	Sequential	80	Contains executable program statements suitable for compilation
Source	Formatted	Sequential	80	Contains "active" card images

(default listing options for the creation or correction run) or the options specified by the user on the MODIFY Control Card. The compile file contains the text portion of all non-directive, active card images. The *DECK and *COMDECK directives are ignored and *CALL directives are replaced with the contents of the comdeck specified. The source file contains the text portion of all active card images. This

includes the *DECK, *COMDECK and *CALL directives which allows the user to specify the source file as input to a MODIFY creation run. This provides a "clean" program library (no correction set information, thereby saving storage space) for the same computer system or allows for easy transport of the program library to another computer system.

III. Operation

This section presents an overview on how to use MODIFY to create a program library and to maintain/retrieve source code stored in a program library.

Library Creation

A one-time-only creation run must establish the program library before any correction runs may be executed. A creation run is implied when MODIFY does not encounter any directives prior to encountering a *DECK or *COMDECK directive.

The user must determine what material will be input and precede each deck with the appropriate *DECK or *COMDECK directive. Data-input cards up to the next *DECK or *COMDECK directive comprise the deck. A common deck must be placed prior to any of the decks calling it.

Each data-input card (including the *DECK or *COMDECK directive) will be assigned an unique identifier in the form of "dname.seqnbr". The "dname" will be the user-assigned name found on the *DECK or *COMDECK directive, and the "seqnbr" is a sequence number assigned by MODIFY, starting with the value of 1 for each new deck introduced.

Library Modifications/Retrievals

Modifications to a program library may include changing the contents of existing deck/comdecks and/or adding new

decks/comdecks to the library. Once all updates are complete, the user may want to simply "retrieve" the contents of an existing deck.

Changes to the contents of existing decks/comdecks are accomplished by a "correction set", a set of directives indicating specific updates and optionally includes data cards which are to be inserted at specified locations. The name of the correction set is assigned by the user via the *IDENT directive. Newly inserted cards will be assigned a "dname.seqnbr" identifier, where "dname" is the ident name. Sequence numbers will be assigned (starting with the value 1 for each ident introduced) as they are inserted into the program library, not necessarily in the order they appear in the correction set. Multiple *IDENT directives may appear in the input file.

Two methods are available to add new decks/comdecks to an already existing program library. In both cases, the "dname" portion of the "dname.seqnbr" identifier will be the name specified on the *DECK or *COMDECK directive. The easiest method is simply to precede each deck with the appropriate *DECK or *COMDECK directive and include this as part of the input file. The decks will be added to the end of the existing program library. Care must be taken, however, to insure that a *DECK or *COMDECK directive is not the first directive of a correction run. Recall that a "creation" run is implied if the first directive encountered is a *DECK or *COMDECK directive.

The second method of insertion can be used if (1) the sole purpose of a correction run is to insert additional decks, (2) the deck is to be inserted in the middle of the existing program library, or (3) the deck to be inserted is located on another file. Each deck must be preceded with the appropriate *DECK or *COMDECK directive and the difference is that an *ADDFILE directive must precede the first *DECK or *COMDECK directive being added. See the "ADDFILE" description, Chapter V, for complete details.

If the purpose of a MODIFY run is simply to retrieve the contents of an existing deck (i.e., no updates desired), specify the "quick" run mode via the MODIFY control card and specify the deck name(s) on a *COMPILE directive.

Card Image Status

As previously explained, the program library deck records (not the directory record) contains multiple card images. A card image is generated for each data-input card in a deck and consists of control information and a text portion. The status of each card image is either "active" or "inactive". All newly generated and RESTOREd card images are "active". Any card images DELETEd, EDITed, FILLed, or YANKed are "inactive". A correction set information designator is added to the control information for a card each time the status of the card is changed. For any one card image, the user may change the status a maximum of twenty times. Only "active" card images are written to

the COMPILE and SOURCE files.

Run Modes

The mode parameter specified on the MODIFY Control Card controls the order in which decks are written to the compile file.

The "quick" mode ("Q" parameter specified) allows *COMPILE directives to specify decks written to the compile file in the order encountered on the library. If the "K" parameter is also specified, the decks are in the order specified on *COMPILE directives.

The "full" mode ("F" parameter specified) ignores *COMPILE directives and updates all decks in the sequence encountered on the library.

If neither "full" nor "quick" is specified, a "normal" mode is implied. The "normal" mode writes to the compile file all decks specified on *COMPILE directives plus decks updated, in the order encountered on the library. If the "K" parameter is specified, the decks are written in the order specified on *COMPILE directives followed by the decks updated.

Overlapping Corrections

An "overlapping correction" occurs when a card image is specified on more than one update-type directive. This may occur when a range of card images is specified on two directives and a card image falls in both ranges. When this

occurs, the directive selected first will be processed while any succeeding directives (specifying a "dname.seqbr" already processed) will be written to the output file along with a message stating that the directive was not processed.

The order of the directives submitted by the user has no bearing on the order in which directives are selected. The order in which directives are selected is first determined by the run mode which selects the processing order of decks. The directives pertaining to a given deck are evaluated and selected in ascending order by the "seqnbr" of the first "dname.seqnbr" identifier on each directive.

Resequencing

After frequent modifications to a program library, it may become necessary or desirable to "clean up" the library. Two methods are available to accomplish this task.

The first method allows specified decks to be resequenced, eliminating all "inactive" card images and renumbering "active" card images starting with the value 1. The drawback of this method is that the library directory is not "cleaned up". That is, ident entries which only apply to a resequenced deck are not removed.

The second method resequences the entire library. The generation of a source file provides a complete set of all "active" card images which may subsequently be specified as the input file to a MODIFY creation run.

Any time decks are resequenced by either method, the history of all changes is lost.

This chapter has provided a general description of how a user might use MODIFY. The next chapter provides a detailed description of the MODIFY Control Card which must be included in the MODIFY input file, and the succeeding chapter provides a detailed description of the MODIFY directives.

IV. Execution

Once the MODIFY program is invoked, the first data-input card on the system-designated input file (i.e., the card reader) must be the MODIFY Control Card. The parameters specified on this card allow the user to (1) change the default file name assigned to a user-accessible file, (2) change the default value of the directive card master control character, (3) specify an alternate input file where the run directives/data cards are to be located, (4) set the run mode, (5) indicate which of the output files are to be generated for the run, and (6) indicate the contents of the output listing file. This chapter provides a detailed description of the MODIFY Control Card and the various optional parameters which are available.

MODIFY Control Card

The MODIFY Control Card must be the first data-input card on the system designated input file. The card must be provided even though all the optional parameters are omitted. The format of the MODIFY card is:

```
MODIFY,p1,p2,...,pn
```

where "p1,p2,...,pn" designates the optional parameters discussed in the next section. The word "MODIFY" must appear in columns 1 through 6, delimiting "MODIFY" and each

parameter with a comma or at least one blank.

Parameters

This section provides the correct format and a description of each optional parameter which may appear on the MODIFY Control Card.

Option	Significance
C - Compile file output	
omitted or C	The compile file will be generated, the contents of which is determined by the run mode and by the directives submitted during the run.
C=filename	The compile file will be generated and identified by the named file.
C=0 (zero)	The compile file is not generated.
F - Full update mode	
F	The full run mode is specified. All known decks are written to the compile file.
omitted	F and Q omitted specifies the normal run mode. The compile file will contain only those decks specified on *COMPILE directives and/or those decks updated during the run.
I - Input file	
omitted or I	The input file will be the system-designated input file (i.e., the card reader).
I=filename	The input file will be the named file.

Option	Significance
K - Compile card sequence	
K	The compile file will contain those decks specified on *COMPILE directives and/or those decks updated, in the order specified by *COMPILE directives followed by decks updated.
omitted	The compile file will contain those decks specified on *COMPILE directives and/or those decks updated, in the order encountered on the library.
L - List options	
L=x	Allows the user to specify the contents of the output file. The "x" may be any one, or any combination, of the list options described in the next section.
omitted	If creation run, L=A12 is automatically selected. If correction run, L=A1234 is automatically selected.
N - New program library output	
N	A new program library will be generated.
omitted	A new program library will not be generated.
N=filename	A new program library will be generated and identified by the named file.
O - Output file	
omitted or O	The output file will be generated on the system-designated output file.
O=filename	The output file will be generated and identified by the named file.

Option	Significance
P - Old program library	
omitted or P	The old program library file is expected to be on the default unit designator and identified by the default file name.
P=filename	The old program library file is expected to be in the named file.
Q - Quick update mode	
Q	The quick run mode is specified. Only decks specified on *COMPILE directives will be written to the compile file.
omitted	Q and F omitted specifies the normal run mode. The compile file will contain only those decks specified on *COMPILE directives and/or those decks updated during the run.
S - Source file	
S	A source file will be generated.
omitted	A source file will not be generated.
S=filename	A source file will be generated on the named file.
* - Master control character	
*=char	The master control character (first character of each directive) for the run is "char".
omitted	The master control character for the run is the default master control character.

List Options

The list ("L") MODIFY Control Card parameter may be used to alter the default list options (creation run is

L=A12, correction run is L=A1234) controlling the contents of the designated output file. The format of the list parameter is :

L=xxx

where "x" designates any of the following options.

Option	Significance
A	Lists known decks/comdecks/ident names and deck names written to COMPILE file
F	All selections other than 0
0	Suppresses all MODIFY listings
1	Lists cards in error and the associated error messages
2	Lists all active MODIFY directives encountered either on input or on the old program library
3	Lists all cards that changed status during MODIFY run
4	Lists all non-MODIFY directives encountered in the input stream
7	Lists all active cards
8	Lists all inactive cards
9	Correction history of all cards listed as a result of list options 7 and 8
B	Three alphabetic lists by deck, comdeck, and ident names

V. Directives

In order to create a program library or modify/retrieve a deck or comdeck in an already existing program library, a set of MODIFY directives must be submitted. The set may consist of one or more directives and a variable number of data-input cards to be inserted into the program library. The MODIFY directives allow the user to (1) identify a correction set which may delete lines of code, insert lines of code, restore lines of code, search and replace lines of code, search and print lines of code, and replace columns in lines of code, (2) identify the beginning of a deck or a common deck, (3) identify common code to be inserted into the contents of a deck or common deck, (4) specify an alternate input file where additional directives and data-input cards may be found, (5) resequence line-of-code identifiers within a deck or comdeck (6) revoke the changes made in a previous run, and (7) retrieve the contents of decks/comdecks suitable for compilation. This chapter provides a detailed description of the fourteen MODIFY directives.

Format

This section describes the general format of the MODIFY directives. All directive cards contain three fields: an identifier field, a name field, and a parameters field. The identifier field contains the master control character

AD-A115 553 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM.(U)
DEC 81 N J MURPHY
UNCLASSIFIED AFIT/GCS/MA/81D-5 NL

2 OF 2
AD
A115 553

END
DATE FILMED
7-82
DTIC

(normally an asterisk) in column 1. The name field starts in column 2 and contains the long or short form of the directive name. The name field must be separated from the parameters field by a comma or by at least one blank. The parameters field contains a variable number of parameters, where a mixture of parameter formats is acceptable, depending on the directive. The following list provides the acceptable formats, a description of each format, and the directives where each format is acceptable:

Format	Description	Directives
dname	The deck or comdeck name; must be one to nine characters in length	*ADDFILE *CALL *COMDECK *COMPILE *DECK *SEQUENCE
idname	The correction set (ident) name; must be one to nine characters in length	*IDENT *YANK
filename	The alternate input file name; character length is machine dependent	*ADDFILE
$dname_a.dname_b$	The deck named $dname_a$ and all decks up to and including $dname_b$, as they appear in the library directory	*COMPILE *SEQUENCE
$idname_a.idname_b$	The ident named $idname_a$ and all idents up to and including $idname_b$, as they appear in the library directory	*YANK

Format	Description	Directives
lname.seqnbr	The deck/comdeck/ident name and the sequence number within the deck/comdeck/ ident; the line-of-code identifier	*DELETE *EDIT *FILL *INSERT *RESTORE *SCAN
/string/	The string of characters to be searched/printed/ replaced, delimited by any character (other than a a comma or a blank) which is not in the string; string must be zero to twenty characters in length	*EDIT *FILL *SCAN
column	The column number where replacement is to start; must be greater than zero and less than 81	*FILL

The parameters within the parameters field must be separated by a comma or at least one blank. The exact placement of each format within a directive card's parameters field is described in detail in the sections to follow.

ADDFILE

```
*ADDFILE,filename,dname
*AF
```

The *ADDFILE directive designates an alternate file where a *DECK or *COMDECK directive and text data may be found (filename), and designates where to insert the information after the specified deck (dname) on the new program library. If "filename" is the primary input file, all cards following the *ADDFILE will be added until a

directive other than a *DECK, *COMDECK, or *CALL directive is encountered. If "filename" is omitted, the primary input file (INPUT or the file specified on the MODIFY Control Card "I=" parameter) is assumed. Only one alternate input file may be specified for each run. That is, if the "I" parameter is used to designate an alternate input filename, the ADDFILE "filename" parameter cannot specify a different file (added decks must be included in the "I=filename" file). If "filename" is omitted and a "dname" specified, two commas must separate the directive name and the "dname" specified. If the add-after deck name "dname" is omitted, the decks will be added to the end of the library. Only one *ADDFILE directive may appear in any one MODIFY run.

Examples:

1) An *ADDFILE is included in file INPUT, decks to be added are immediately following the *ADDFILE, and the information is to be added to the end of the library.

```
*ADDFILE,INPUT
*ADDFILE
```

2) An *ADDFILE is included in file INPUT, decks to be added are in file NEWFILE, and the information to be added is to be inserted after deck name DK12.

```
*ADDFILE,NEWFILE,DK12
```

3) An *ADDFILE is included in the input file NEWFILE (i.e., I=NEWFILE), decks to be added are immediately following the *ADDFILE, and the information to be added is to be inserted after deck name DK12.

```
*ADDFILE,NEWFILE,DK12
*ADDFILE,,DK12
```

CALL

```
*CALL,dname
*CA
```

The *CALL directive designates a comdeck "dname", the contents of which is written to file COMPILE (excluding the *COMDECK card) in place of the *CALL directive. Common decks may call other common decks. However, a common deck must not call itself or call decks that contain calls to the common deck. A *CALL is the only directive which may appear within the contents of a deck or comdeck. It is treated as a data card of the deck in which it appears.

Example:

A program library contains one deck, DK1, consisting of a FORTRAN 77 program and two comdecks, COM1 and COM2, consisting of FORTRAN 77 COMMON statements:

```
*DECK,DK1                          *COMDECK,COM1
      PROGRAM PROG1                      COMMON/INT/I1,M
*CALL,COM1                               COMMON/REAL/X,Y,Z
*CALL,COM2
      COMMON/DATA/COUNT,LENGTH
      M=1                          *COMDECK,COM2
        .                                COMMON/CHAR/NAME
        .
        .
      END
```

The contents of DK1, as it would appear in file COMPILE, follows:

```
PROGRAM PROG1
COMMON/INT/I1,M
COMMON/REAL/X,Y,Z
COMMON/CHAR/NAME
COMMON/DATA/COUNT,LENGTH
M=1
  .
  .
  .
END
```

COMDECK

```
*COMDECK,dname
*CD
```

The *COMDECK directive designates the name of a new common deck "dname" and denotes its beginning. The contents of a common deck is written to file COMPILE only as a result of a *CALL directive encountered in a deck or another comdeck. The "dname" must differ from any deck or comdeck names already in the library directory. The data cards following the *COMDECK are considered part of the common deck until a directive other than a *CALL is encountered. No more than 999,999 data cards may be introduced by one *COMDECK. A common deck must be introduced before the decks or comdecks calling it are introduced.

COMPILE

*COMPILE,$dname_1$,$dname_2$,$dname_3$,...,$dname_n$
*C

*COMPILE,$dname_a$.$dname_b$
*C

The *COMPILE directive designates the name of the deck "dname" to be written to file COMPILE. The first form requests that one or more decks be written to file COMPILE. The second form requests that the deck "$dname_a$" and all decks up to and including "$dname_b$" be written to file COMPILE, as they appear on the library directory. The two

types of formats may appear on the same *COMPILE. Any deck introduced or any deck updated during the run will automatically be written to file COMPILE. If "dname" is a new deck name, the *DECK directive introducing "dname" and its contents must precede the *COMPILE. The order of the decks written to file COMPILE depends on the run mode (full, quick, or normal) and the "K" parameter specified on the MODIFY Control Card. No more than 300 deck names may be specified (directly and/or indirectly) on one MODIFY run.

Examples:

The program library directory currently contains seven decks: DK1, DK2, DK3, DK4, DK5, DK6, and DK7.

1) Write the contents of DK3 and DK4 to file COMPILE.

```
*COMPILE,DK3,DK4
*COMPILE,DK4,DK3
*COMPILE,DK3.DK4
```

2) Write the contents of all the decks except DK4 to file COMPILE.

```
*COMPILE,DK1,DK2,DK3,DK5,DK6,DK7

*COMPILE,DK1,DK2,DK3
*COMPILE,DK5,DK6,DK7

*COMPILE,DK1,DK3,DK5.DK7

*COMPILE,DK1.DK3,DK5,DK6,DK7
```

DECK

```
*DECK,dname
*DK
```

The *DECK directive designates the name of a new deck "dname" and denotes its beginning. The "dname" must differ

from any of the deck or comdeck names already in the library directory. The data cards following the *DECK are considered part of the deck until a directive other than a *CALL is encountered. No more than 999,999 data cards may be introduced by one *DECK.

DELETE

```
*DELETE,lname.seqnbr
*D

*DELETE,lname.seqnbr_a,lname.seqnbr_b
*D
```

The *DELETE directive designates the card, or a range of cards, to be deactivated. Any data cards following the *DELETE will be inserted following the deactivated card(s) on the NEWPL and OUTPUT files and will replace the deactivated card(s) on the COMPILE and SOURCE files. The insertion process will continue until a directive other than a *CALL is encountered. The first form requests that one card, one line of code, be deactivated. The second form requests that the card "lname.seqnbr$_a$" and all cards up to and including "lname.seqnbr$_b$" be deactivated. The range may include cards previously deactivated. The *DELETE does not actually delete cards but flags them as inactive. The cards retain the same "lname.seqnbr" identifier and may be referred to in the same way on a succeeding run. The "lname" may be a deck name, a common deck name, or an ident name. The "seqnbr" may not exceed 999,999. Each card's

"lname.seqnbr" identifier will be printed with its text on the output listing file.

Examples:

A partial listing of the deck DK1 follows, where a previous correction set IDENT1 made changes as indicated by the line designator:

```
*DECK,DK1                                DK1          1
      PROGRAM PROG1                      DK1          2
      INTEGER TABLE(8,1000)              IDENT1      10
      M=1                                DK1          3
      READ *,(TABLE(J,M),J=1,8)          DK1          4
1     IF (TABLE(1,M).GE.0) THEN          DK1          5
         M=M+1                           IDENT1      11
         READ *,(TABLE(J,M),J=1,8)       DK1          7
         GO TO 1                         DK1          8
      ENDIF                              IDENT1      12
```

1) Delete the priming READ statement.

```
*DELETE,DK1.4
```

2) Delete all lines associated with the IF statement.

```
*DELETE,DK1.5,IDENT1.12
```

EDIT

```
*EDIT,/string1/,/string2/,lname.seqnbr
*ED

*EDIT,/string1/,/string2/,lname.seqnbra,lname.seqnbrb
*ED
```

The *EDIT directive designates that a string of characters "$string_1$" be replaced by a string of characters "$string_2$" on the card "lname.seqnbr", or on the range of cards from "$lname.seqnbr_a$" to "$lname.seqnbr_b$". If "$string_1$" is not found, no replacement takes place. The strings,

"string$_1$" and "string$_2$", may contain zero to twenty characters and must be delimited by a character (other than a comma or a blank) which is not in the string. The delimiter used for "string$_1$" may be different from the delimiter used for "string$_2$". Each card in the specified range which contains "string$_1$" is deactivated and a new card containing "string$_2$" in place of "string$_1$" is created. If the strings are of different lengths, the characters on the created card are moved appropriately. A warning message will be issued if the moving of characters causes the text to exceed 72 characters.

Examples:

1) Replace the characters "CONTINE" with "CONTINUE" in line 10 of the deck DK1.

```
*EDIT,/CONTINE/,$CONTINUE$,DK1.10
```

2) Replace the characters "FI" with "IF" in all 50 lines of code in the deck DK1.

```
*EDIT,/FI /,/IF /,DK1.1,DK1.50
```

FILL

```
*FILL,column,/string/,lname.seqnbr
*FI

*FILL,column,/string/,lname.seqnbr_a,lname,seqnbr_b
*FI
```

The *FILL directive designates that a string of characters "string" be placed in the card "lname.seqnbr", or in the range of cards "lname.seqnbr$_a$" to "lname.seqnbr$_b$",

starting in the column number "column". "String" may contain zero to twenty characters and must be delimited by a character (other than a comma or a blank) which is not in the string. Each card in the specified range is deactivated and a new card containing "string" starting in "column" is created. A warning message will be issued if the replacement of "string" causes the text to exceed 72 characters.

Examples:

1) Place a "C" in column 1 of the DK1 lines 10 through 15 so that the lines will be treated as comments.

```
*FILL,1,/C/,DK1.10,DK1.15
```

2) Remove the "C" from column 1 of the DK1 lines 10 through 15 so that the lines will again be executed.

```
*FILL,1,/ /,DK1.10,DK1.15
```

IDENT

```
*IDENT,idname
*ID
```

The *IDENT directive designates the name of a correction set "idname" and denotes its beginning. The "idname" must differ from any of the ident names already in the library directory. The "idname" name remains in effect until another *IDENT is encountered. Any card inserted by "idname" will be given an identifier in the form of "idname.seqnbr". Sequencing of new cards starts with one for each ident. Any card introduced by an *IDENT may be

changed up to a maximum of 19 times by succeeding *IDENT directives. Any card introduced by a *DECK or *COMDECK may be changed up to a maximum of 20 times by succeeding *IDENT directives.

INSERT

```
*INSERT,lname.seqnbr
*I
```

The *INSERT directive designates the card "lname.seqnbr" after which the data cards following the *INSERT will be inserted. The insertion process will continue until a directive other than a *CALL is encountered. The cards inserted will be given an identifier in the form of "idname.seqnbr", where "idname" is the name on the *IDENT introducing the correction set.

RESTORE

```
*RESTORE,lname.seqnbr
*R

*RESTORE,lname.seqnbr_a,lname.seqnbr_b
*R
```

The *RESTORE directive designates the card, or a range of cards, to be reactivated. Any data cards following the *RESTORE will be inserted into COMPILE and NEWPL, following the reactivated card(s). The insertion process will continue until a directive other than a *CALL is

encountered. The first form requests that one card be reactivated. The second form requests that the card "lname.seqnbr$_a$" and all cards up to and including "lname.seqnbr$_b$" be reactivated. The range may include cards already active.

SCAN

*SCAN,/string$_1$/,/string$_2$/,lname.seqnbr
*SC

*SCAN,/string$_1$/,/string$_2$/,lname.seqnbr$_a$,lname.seqnbr$_b$
*SC

The *SCAN directive designates tnat a string of characters "string$_1$" be replaced temporarily by a string of characters "string$_2$" on the card "lname.seqnbr", or on the range of cards from "lname.seqnbr$_a$" to "lname.seqnbr$_b$". The strings "string$_1$" and "string$_2$" may contain zero to twenty characters and must be delimited by a character (other than a comma or a blank) which is not in the string. The delimiter used for "string$_1$" may be different from the delimiter used for "string$_2$". Each card in the specified range which contains "string$_1$" is not permanently changed. The text is used to create a temporary card containing "string$_2$" in place of "string$_1$" and the temporary card is written to the output listing file. The *SCAN does not permanently deactivate nor create cards. The purpose of this directive is to verify the effects of the string replacements before the changes are actually made with the *EDIT directive.

SEQUENCE

```
*SEQUENCE,dname1,dname2,...,dnamen
*S

*SEQUENCE,dnamea.dnameb
*S
```

The *SEQUENCE directive designates the deck or comdeck "dname" to be resequenced. The first form requests that one or more decks be resequenced. The second form requests that the deck or comdeck "$dname_a$" and all decks and comdecks up to and including "$dname_b$" be resequenced. All active cards in "dname" are resequenced with the identifier "dname.seqnbr". All history-type control information is deleted from active cards and all inactive cards are purged, thereby "cleaning up" the contents of the library record(s). The *SEQUENCE does not remove ident names from the directory, even though all "idname.seqnbr" designators have been purged.

YANK

```
*YANK,idname1,idname2,...,idnamen
*Y

*YANK,idnamea.idnameb
*Y
```

The *YANK directive designates the name of the correction set "idname" and causes the effects of the correction set to be reversed. That is, any card introduced

by the ident "idname" is deactivated and any card deactivated by "idname" is reactivated. The first form requests that one or more idents be yanked. The second form requests that the ident "$idname_a$" and all idents up to and including "$idname_b$" be yanked, as they appear on the library directory. The two types of formats may appear on the same *YANK.

VI. VAX/Cyber Job Control Examples

This chapter provides four examples of job control streams which could be used to invoke MODIFY on the Avionics Laboratory's VAX computer system and the Aeronautical System Division's Cyber computer system. The first two examples illustrate the job control language used for DEC VAX 11/780 creation and correction runs. The last two examples illustrate the job control language used for CDC Cyber 74/750 creation and correction runs.

VAX Creation Run

```
$ JOB MURPHY
$ PASSWORD xxxxxx
$ RUN MODIFY
MODIFY,N

   (Creation directives and source code cards)

$ EOJ
```

VAX Correction Run

```
$ JOB MURPHY
$ PASSWORD xxxxxx
$ RUN MODIFY
MODIFY,N

   (Update directives and optional source code
    cards)

$ EOJ
```

Cyber Creation Run

```
NJM,T20.       T820077,MURPHY,55533.
REQUEST,NEWPL,*PF.
ATTACH,MODIFY,MODIFYBINARY.
MODIFY.
CATALOG,NEWPL,PROGRAMLIBRARY.
7/8/9
MODIFY,N

   (Creation directives and source code cards)

6/7/8/9
```

Cyber Correction Run

```
NJM,T20.       T8200077,MURPHY,55533.
REQUEST,NEWPL,*PF.
ATTACH,MODIFY,MODIFYBINARY.
ATTACH,OLDPL,PROGRAMLIBRARY.
MODIFY.
CATALOG,NEWPL,PROGRAMLIBRARY.
7/8/9
MODIFY,N

   (Update directives and optional source code
    cards)

6/7/8/9
```

Appendix D

MODIFY Functional Charts

Appendix D

MODIFY Functional Charts

This appendix provides an overview, first-level, and second-level Structured Analysis and Design Technique (SADT) functional charts. The SADT charts do not show housekeeping functions, and they must show at least three and no more than six functional areas on each chart.

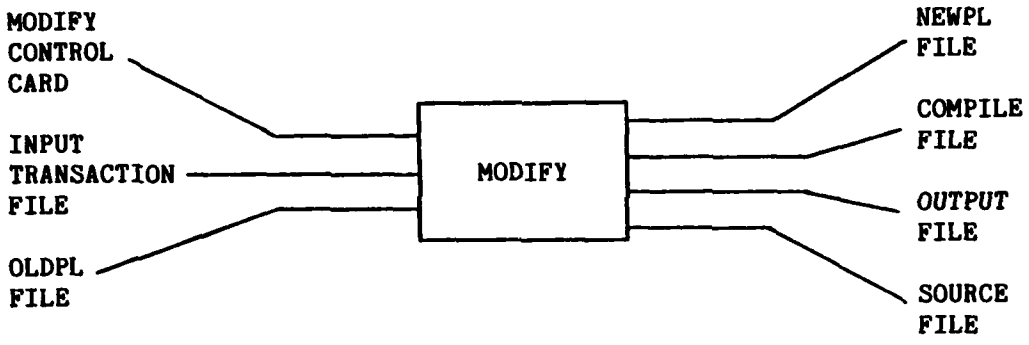


Figure D-1. MODIFY Program Overview (A-0)

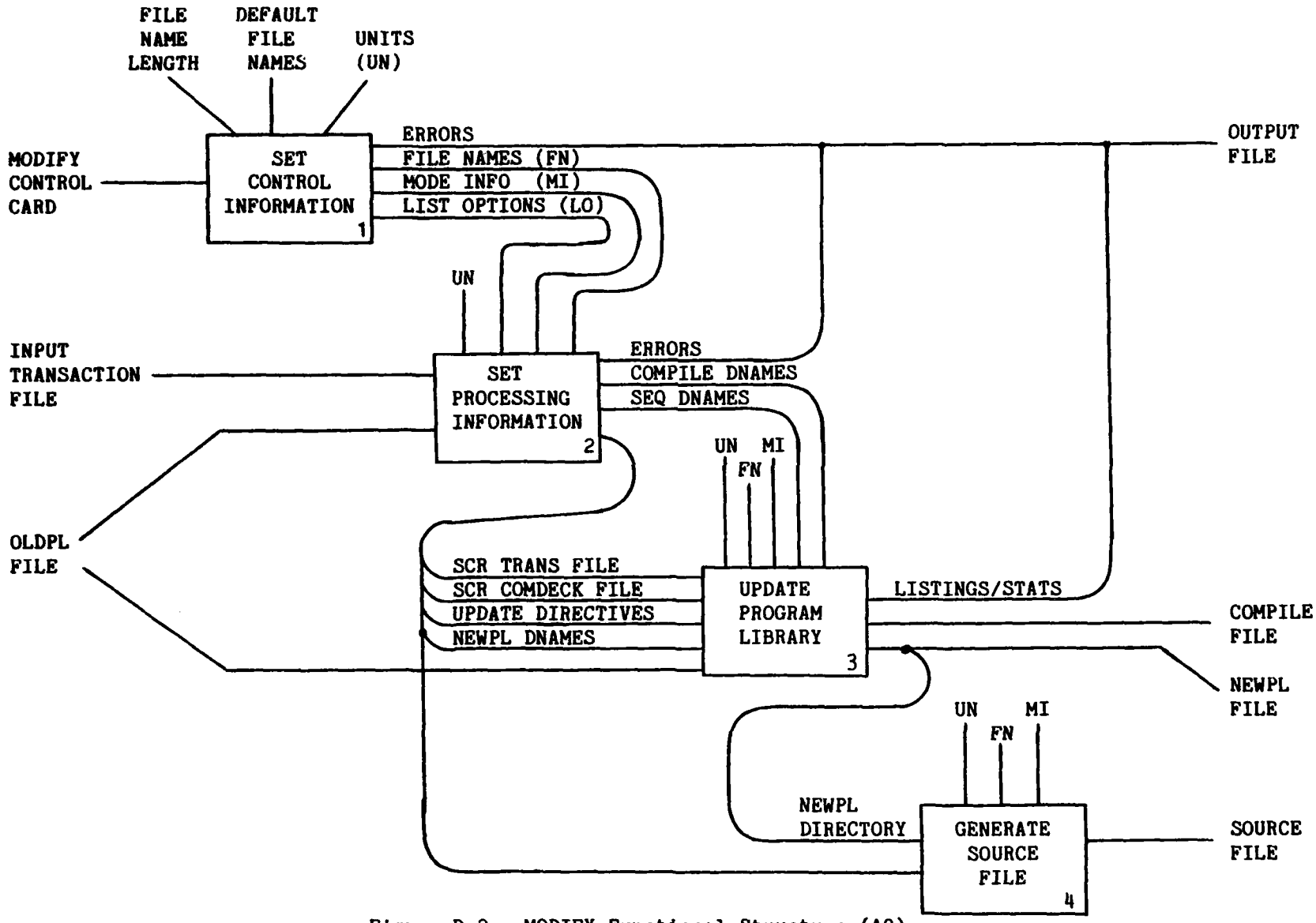


Figure D-2. MODIFY Functional Structure (A0)

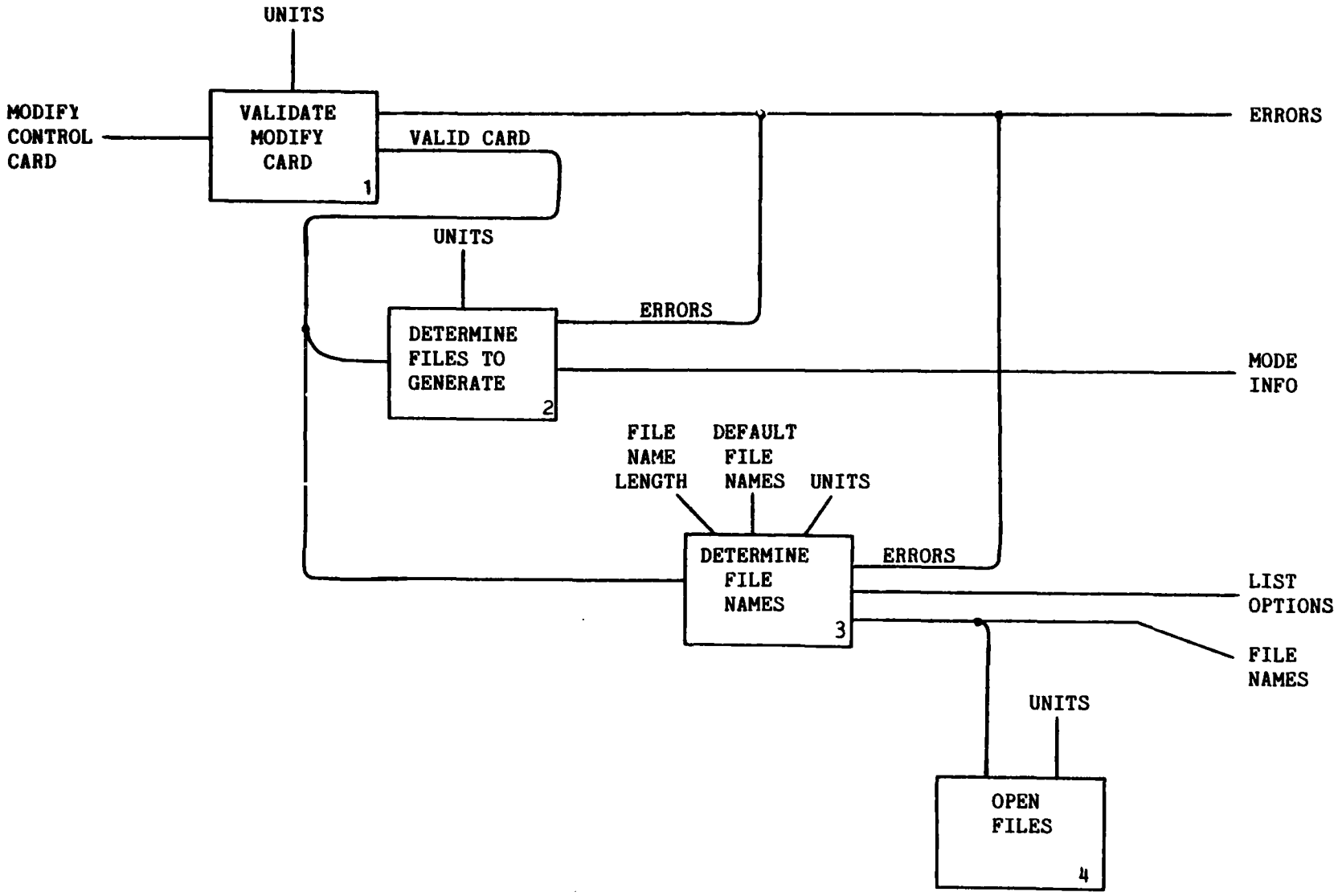


Figure D-3. Set Control Information (A1)

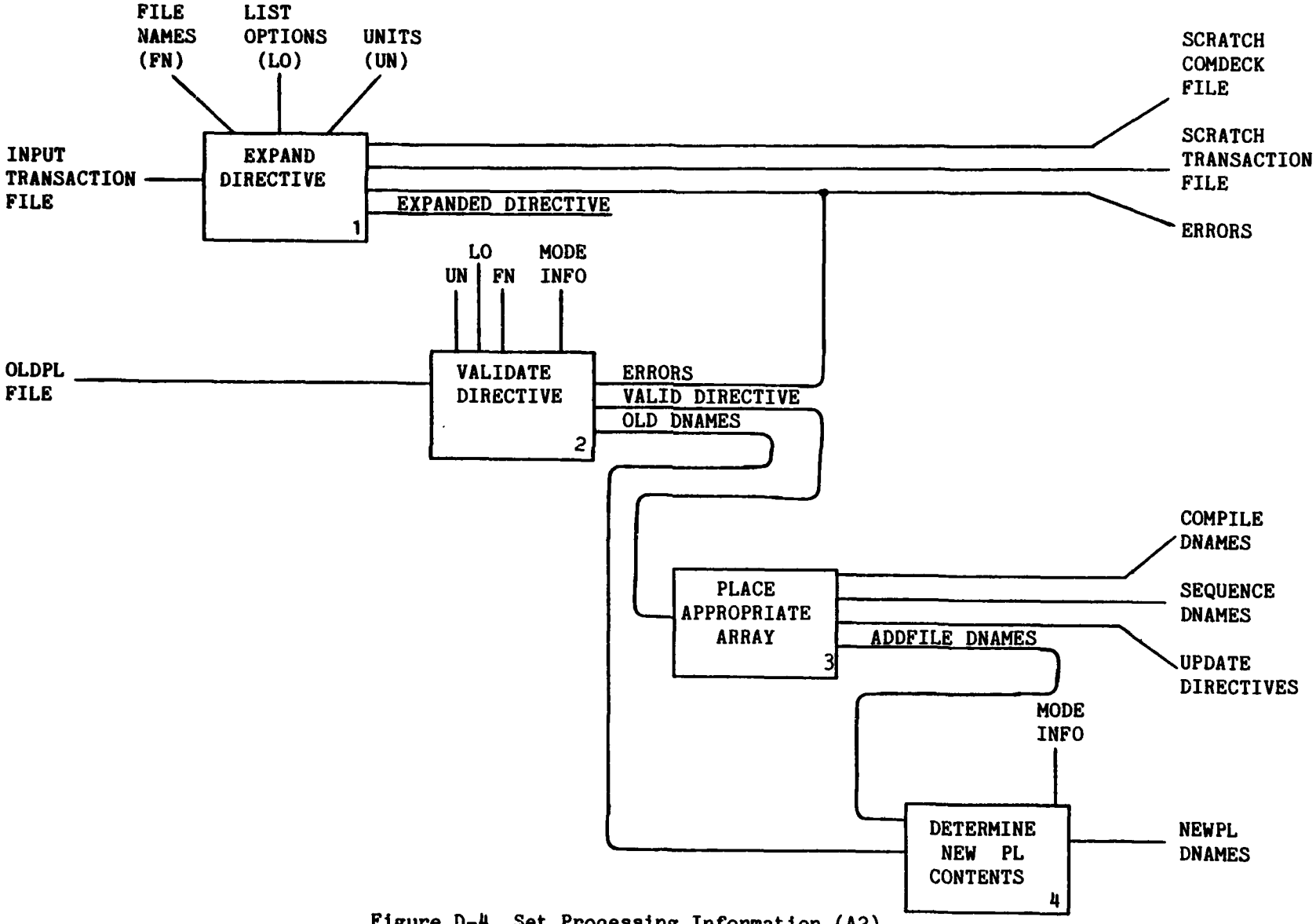


Figure D-4. Set Processing Information (A2)

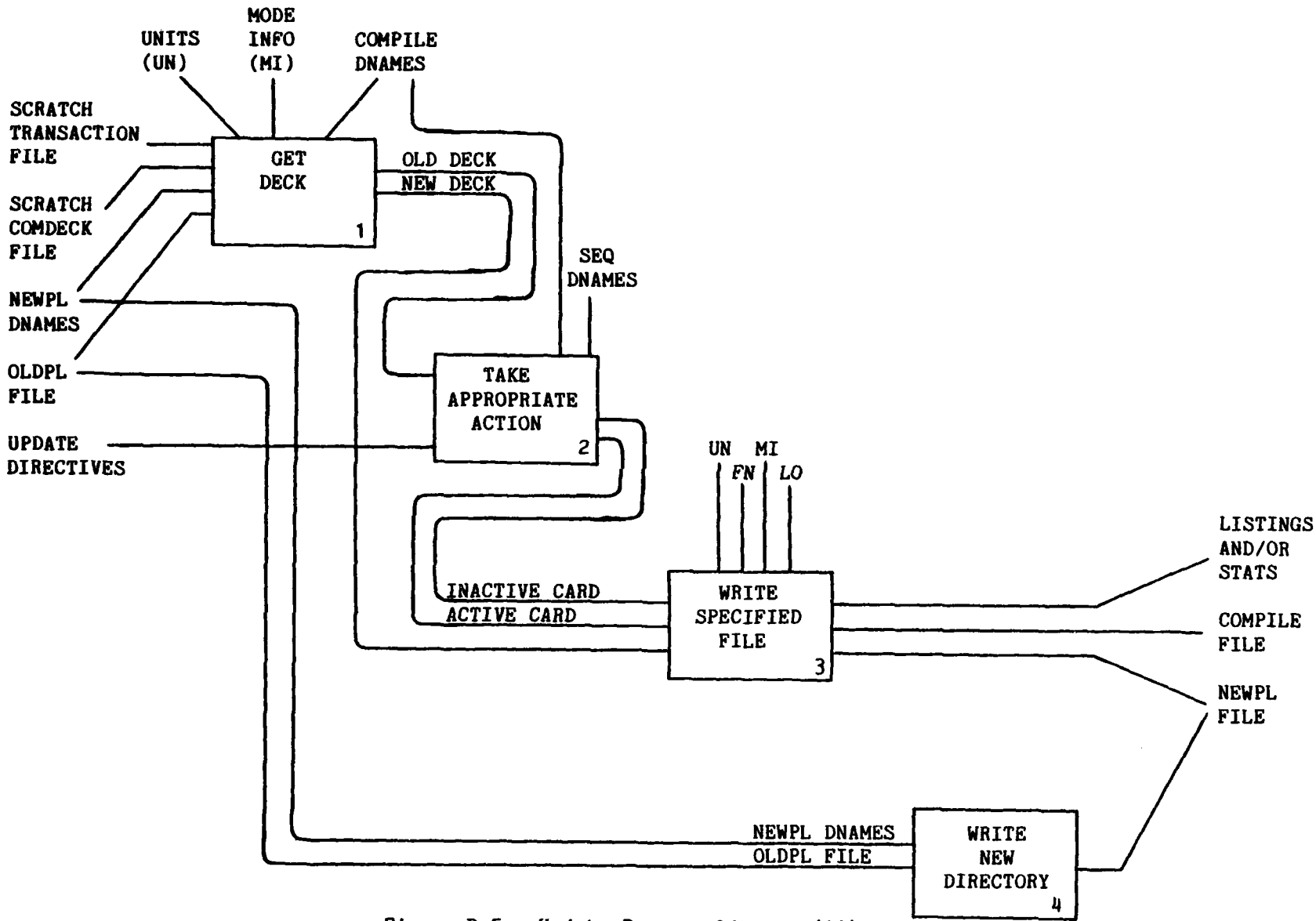


Figure D-5. Update Program Library (A3)

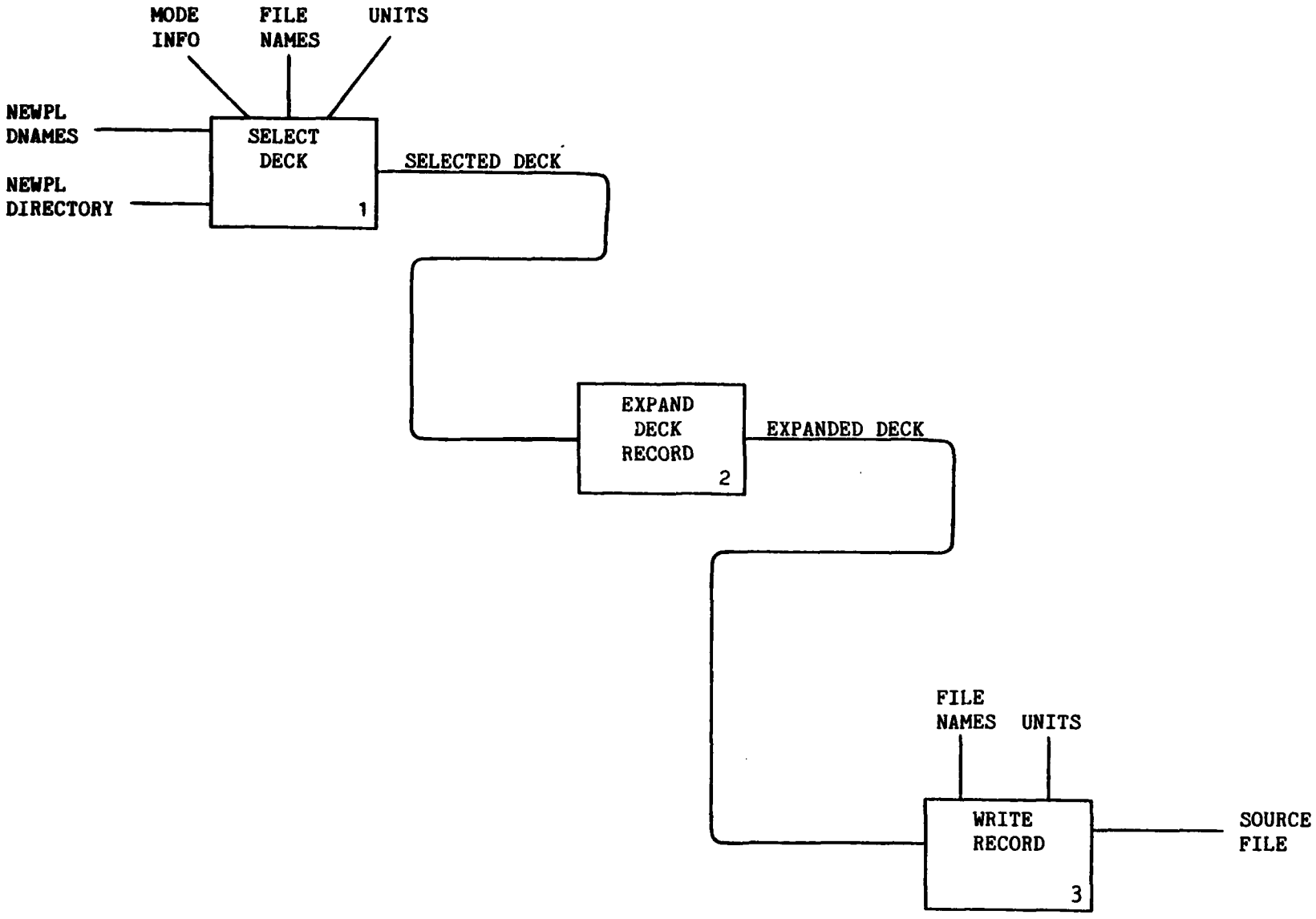


Figure D-6. Generate Source File (A4)

Appendix E

Description of MODIFY's Subroutines

Appendix E

Description of MODIFY's Subroutines

The MODIFY program consists of a main program and 54 subroutines. This appendix provides a brief description of each one of the 54 subroutines. The list of subroutines is in alphabetical order by the subroutine name used in the MODIFY program. Standard FORTRAN 77 requires that subroutine names be less than or equal to six characters in length; therefore, a more descriptive title is given at the beginning of each definition.

BLDCOM - BUILD COMPILE is used when a deck is to be retrieved (no updates) and written to file COMPILE. Each active card image is written to the compile and scratch output files.

BLDCON - BUILD CONTROL INFORMATION retrieves card image words and places the contents of the words in the appropriate variable names/arrays.

BLDNEW - BUILD NEWNAM ARRAY merges the contents of "ADDFIL" and "OLDNAM" arrays.

BLDSOU - BUILD SOURCE FILE takes all comdeck records followed by deck records and writes active card images to the source file.

CHGNAM - CHANGE NAME retrieves the file name or list options specified as the value of a parameter on the MODIFY control card and updates the FILES or LIST array.

CHKYAN - CHECK YANK scans each card images's correction-set-information words and reverses the effects of the ident name specified on *YANK directive(s).

COMPAC - COMPACT takes a record and determines the number of spaces preceding the text, the number of text characters, and returns the text characters left justified.

EXPAND - EXPAND takes the directive abbreviation found and returns the full directive name.

FIND - FIND is used to locate a given name in a given array. If found, the array entry is returned.

FINNEW - FINISH NEWPL writes any decks not already processed to the new program library.

GENCRD - GENERATE CARD IMAGE controls the building of the NEWREC array (a new program library record) by placing n-words in the array for each card image and starting a continuation record if necessary.

GENDIR - GENERATE DIRECTORY builds and writes the new program library directory using information from the old directory and the NEWNAM, NEWCON, and OLDCON arrays.

GENSOU - GENERATE SOURCE FILE controls the building of the source file by selecting all active card images in comdecks followed by all active card images in decks.

GET - GET is used to retrieve text characters which have been stored in a integer record on the unformatted, direct-access program library.

GETDEC - GET DECK selects the next NEWNAM deck/comdeck entry to process, based on the run mode (full, quick or normal).

GETLST - GET LIST sets the entries in the logical array LIST according to the values found after the "L" parameter on the MODIFY Control Card.

GETNAM - GET NAME scans a string of characters and returns the name found.

GETOLD - GET OLD DIRECTORY brings in the OLDPL directory and builds the arrays OLDNAM and OLDCON.

INIT - INITIALIZE sets the input/output file units, sets the file names with the user-supplied default file names, and writes the header to the output file.

INSERT - INSERT takes text characters and places them in the specified *n*-integer words.

INSNEW - INSERT NEW controls the generation of new data cards to be placed in specified deck/comdeck.

NEWENT - NEW ENTRY takes new deck/comdeck/ident names and places appropriate information in the NEWREC array.

NFIND - NUMERIC FIND takes the integer variable specified and determines if the value is in the specified array. If found, returns the array entry.

NMODE - NORMAL MODE selects the next deck to be processed with precedence being given to those decks referenced on *COMPILE directives followed by those decks which are to be updated.

OLDENT - OLD ENTRY is used in generating a new PL directory using the appropriate information in NEWREC to build the new directory.

OPNFIL - OPEN FILES opens the appropriate files as specified by the array FILES.

PRIME - PRIME performs a priming read on the input transaction file and determines if the run is to be creation or correction.

PRINT - PRINT controls all lines written to the output file by assigning carriage control, writing the header on top of each form, and insuring no more than MAXLIN lines are printed per form.

PRODAT - PROCESS DATA CARD increments the appropriate counter and writes the data card to either the scratch input file or the scratch comdeck file.

PRODEL - PROCESS DELETE DIRECTIVE deactivates active card images and writes the text to the output file.

PRODIR - PROCESS DIRECTIVE validates input directives and places them in the appropriate array.

PROEDI - PROCESS EDIT DIRECTIVE scans the text for the given substring, deactivates the card image, and builds a new card image with the substituted substring.

PROFIL - PROCESS FILL DIRECTIVE places the substring on the text where specified, deactivates the old card image, and builds a new card image with substituted substring.

PROMOD - PROCESS MODIFY CONTROL CARD scans for the MODIFY parameters and sets the array MODE appropriately.

PRONEW - PROCESS NEW locates the new deck/comdeck in the scratch input file and controls the writing of COMPILE, NEWPL, scratch output, and OUTPUT files.

PRORES - PROCESS RESTORE DIRECTIVE reactivates inactive card images and writes the text to the output file.

PROSCA - PROCESS SCAN DIRECTIVE scans the record for the substring, builds a new image with substituted substring, and writes the new image to the output file (no updating).

QMODE - QUICK MODE selects the next deck to be processed with precedence being given to those decks referenced on *COMPILE directives.

SEQDK - SEQUENCE DECK takes each active card image in the deck/comdeck and sequentially renumbers the card images, controls the writing of COMPILE, NEWPL, scratch output, and OUTPUT files.

SETCNT - SET COUNTER is part of processing the input transaction file. If a *CALL directive or a data-input card is realized, a counter is incremented to indicate another data card in the deck. Other directives cause the logical array TYPE to be set to false, indicating a new counter will be started.

SETCON - SET CONTROL INFORMATION is used to process the MODIFY Control Card and opens appropriate files.

SETPRO - SET PROCESSING INFORMATION is used to read and process the input transaction file (directives and data-input cards).

SETUP - SETUP prepares the selected deck/comdeck for processing by building arrays DECUPD and YANK, and setting variables DNAME and DKINFO.

UPDATE - UPDATE looks at each card image in the selected deck/comdeck, performs any changes requested by update directives, and controls the writing of the COMPILE, NEWPL, scratch output, and OUTPUT files.

UPDPL - UPDATE PROGRAM LIBRARY controls the creating/retrieving/updating of the program library.

VALAF - VALIDATE ADDFILE DIRECTIVE insures proper syntax, validates parameters, and updates ADDAFT and FILES.

VALCA - VALIDATE CALL DIRECTIVE insures that the comdeck name exists and increments the appropriate counter.

VALCDI - VALIDATE COMDECK/DECK/IDENT DIRECTIVES insures the name does not already exist, sets TYPE array, and increments a counter if directive is a *DECK or *COMDECK.

VALCSY - VALIDATE COMPILE/SEQUENCE/YANK DIRECTIVES insures name(s) exist and places the name(s) in appropriate arrays (either COMPIL, RESEQ, or YANK).

VALREM - VALIDATE REMAINING DIRECTIVES (DELETE, INSERT, RESTORE, EDIT, SCAN, and FILL) insures the DNAME.SEQ exists, places the directive in the UPDDIR array, and writes the directive and any data cards which follow to the scratch input file.

WRAPUP - WRAP UP writes the scratch output listing to the output file, generates list option "B", and informs the user of directory utilization.

WRICOM - WRITE COMPILE FILE expands any *CALL directives and writes all records to file COMPILE.

WRICRD - WRITE CARD IMAGE controls the writing of all card images, checking the status of the card image and which files are to be generated.

WRINEW - WRITE NEWPL determines the REC number of the record to be written and writes the NEWREC array to the NEWPL file.

Vita

Nancy Joan (Keller) Murphy was born on 20 July 1948 in Piqua, Ohio. She graduated from Newton Township Local High School, Pleasant Hill, Ohio, in 1966. After working four years in Columbus, Ohio, she enlisted in the USAF and was eventually assigned to Charleston AFB, South Carolina, as an Air Operations Specialist. In September 1974, she attended Wright State University, Dayton, Ohio, under the Airman Education and Commissioning Program and received her Bachelor of Science Degree in Quantitative Business Analysis. Upon completion of Officers Training School in March 1977, she was assigned to Headquarters, Air Force Logistics Command, Wright-Patterson AFB, Ohio. In June 1980, she entered the School of Engineering, Air Force Institute of Technology to pursue her Masters Degree.

Permanent address: 1115 Wayne Street
Troy, Ohio 45373

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/MA/81D-5	2. GOVT ACCESSION NO. AD-A115 553	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MODIFY: A MACHINE-INDEPENDENT MAINTENANCE PROGRAM		5. TYPE OF REPORT & PERIOD COVERED MS THESIS 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) NANCY J. MURPHY Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory, Electronic Warfare Division (AFWAL/AAWA) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE December 1981 13. NUMBER OF PAGES 119
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

15 APR 1982

18. SUPPLEMENTARY NOTES

Approved for public release IAW AFR 190-17

FREDERIC C. LYNCH, Major, USAF
Director of Public Affairs

Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Maintenance
ADP Utility
Library Maintenance
Batch Utility
Batch Editor
Configuration Management Control

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

MODIFY is a machine-independent batch utility designed to assist programmers in maintaining source files. MODIFY is written in standard FORTRAN 77. MODIFY handles routine update/retrieval functions and provides a complete audit trail of changes.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)